RICARDO HILLBRECHT

A VIRTUAL-MACHINES-MIB: UMA ABORDAGEM BASEADA EM SNMP PARA O GERENCIAMENTO DE MÁQUINAS VIRTUAIS

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: LUÍS CARLOS ERPEN DE BONA

CURITIBA

2012

RICARDO HILLBRECHT

A VIRTUAL-MACHINES-MIB: UMA ABORDAGEM BASEADA EM SNMP PARA O GERENCIAMENTO DE MÁQUINAS VIRTUAIS

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: LUÍS CARLOS ERPEN DE BONA

CURITIBA

2012

RICARDO HILLBRECHT

A VIRTUAL-MACHINES-MIB: UMA ABORDAGEM BASEADA EM SNMP PARA O GERENCIAMENTO DE MÁQUINAS VIRTUAIS

Dissertação aprovada como requisito parcial à obtenção do grau de Mestre no Programa de Pós-Graduação em Informática da Universidade Federal do Paraná, pela Comissão formada pelos professores:

Orientador: LUÍS CARLOS ERPEN DE BONA
Departamento de Informática, UFPR

Prof. Dr. Eduardo Todt
Departamento de Informática, UFPR

Prof. Dr. Magnos Martinello
Departamento de Informática, UFES

Curitiba, 15 de outubro de 2012

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Este trabalho apresenta a *Virtual-Machines-MIB*, uma MIB para a gerência de máquinas virtuais baseada no *Simple Network Management Protocol* (SNMP). A *Virtual-Machines-MIB* define uma interface padronizada para a gerência de máquinas virtuais, permitindo que a mesma ferramenta possa gerenciar, através do protocolo SNMP, diferentes monitores de máquinas virtuais, como KVM, Xen e VMWare. Diferente da maior parte das MIB's existentes, a *Virtual-Machines-MIB* permite ao gerente não apenas monitorar aspectos da máquina física e das VM's, mas também executar ações de controle, como criar, excluir, reiniciar, ligar, desligar e congelar VM's. Também é possível alterar o nome, a quantidade de memória RAM e de CPU's das VM's, além de alterar as unidades de armazenamento das mesmas. Resultados práticos são apresentados utilizando ferramentas de gerência SNMP comuns para gerenciar diferentes monitores de máquinas virtuais. Para isso, foram criados agentes SNMP que oferecem suporte à *Virtual-Machines-MIB* e instalados em máquinas com o KVM e o Xen. Os agentes foram criados com base no agente SNMP de domínio público NET-SNMP, que foi estendido para que passe a oferecer suporte à *Virtual-Machines-MIB* utilizando as funções da *libvirt*.

ABSTRACT

This work presents *Virtual-Machines-MIB*, a MIB (Management Information Base) directed to virtual machines management through SNMP (Simple Network Management Protocol). *Virtual-Machines-MIB* aims to define a standard interface for virtual machines management, allowing the management of several virtual machines monitors, like Xen, KVM and VMWare, with a common SNMP management tool. Different from most existing MIBs, which allows the manager to perform only monitoring operations, *Virtual-Machines-MIB* allows to perform control operations, like create, delete, restart, turn on, pause and shut down virtual machines. It is also possible to use the proposed solution to change a virtual machine's name, amount of RAM, virtual CPU's and virtual storage drives. Practical results are presented using ordinary SNMP management tools performing KVM and Xen management. To do this, SNMP agents which support *Virtual-Machines-MIB* were developed and installed on KVM and Xen hosts. These SNMP agents are based on NET-SNMP public domain's agent, that was extended to support *Virtual-Machines-MIB* using *libvirt* API.

CAPÍTULO 1

INTRODUÇÃO

A virtualização é uma tecnologia que permite que mais de uma instância de sistema operacional seja executada em uma mesma máquina física ao mesmo tempo. Uma camada de virtualização oferece suporte em baixo nível para a criação de múltiplas máquinas virtuais ou VM's (*Virtual Machines*), que são independentes e isoladas umas das outras. Essa camada de virtualização é chamada de monitor de máquinas virtuais ou VMM (*Virtual Machine Monitor*) [41]. O uso de máquinas virtuais simplifica o gerenciamento dos centros de dados, aumenta a eficiência dos recursos e a confiabilidade nos serviços [24].

Recentemente houve o surgimento e a popularização da computação nas nuvens, ou *cloud computing*, um novo paradigma de organização e entrega de serviços através da Internet. A virtualização constitui a base da computação nas nuvens, uma vez que oferece a capacidade de agregar os recursos computacionais de diversos *clusters* de máquinas físicas e atribuir dinamicamente recursos virtuais para as aplicações com base na demanda [55]. Devido à difusão do uso de computação nas nuvens, aliado ao crescimento da quantidade de VM's, surgiram novos desafios para a gerência do ambiente computacional [29, 24], assim como para a gerência de VM's.

Existem poucos sistemas de gerência de VM's capazes de suportar mais de um VMM, pois os fabricantes mantém a gerência das VM's restrita aos próprios sistemas de gerência por razões comerciais [29]. Os principais VMM's existentes no mercado, como Xen [22], VMWare [20] e Hyper-V [11], fornecem aplicativos próprios de gerência, como o *Citrix Xen Center*, *VMWare Virtual Center* e *Microsoft Virtual Machine Manager*, que são capazes de gerenciar somente o VMM correspondente. Existem também sistemas direcionados à gerência de ambientes de computação nas nuvens, como o Eucalyptus [42] e o OpenNebula [49], que são capazes de gerenciar mais de um VMM para atender aos requisitos de um provedor de computação nas nuvens. No entanto tais sistemas estão limitados a

determinados VMM's, não sendo possível utilizá-los para gerenciar um VMM que não seja compatível com os mesmos.

Diversos fatores, entre eles a computação nas nuvens, estão impulsionando a integração de infraestruturas virtuais, no entanto nem sempre é possível gerenciar de forma integrada dois ou mais VMM's diferentes pois as interfaces de gerência que os mesmos oferecem não são uniformes. Uma interface padronizada para a gerência de máquinas virtuais permitiria a criação de ferramentas de gerência com a capacidade de interagir com diversas plataformas de virtualização [43]. Caso os VMM's passassem a oferecer uma interface padronizada de gerência, sistemas como o Eucalyptus e o OpenNebula também poderiam ser beneficiados, pois poderiam gerenciar as VM's através da interface padronizada e assim interagir com qualquer VMM.

O *Simple Network Management Protocol* (SNMP) [25] é a arquitetura de gerência de redes mais amplamente utilizada. A arquitetura do SNMP baseia-se nos seguintes elementos: agente de gerência, estação de gerência, protocolo de gerência e base de informações de gerência ou *Management Information Base* (MIB). A MIB é composta de objetos de gerência que armazenam as informações relativas aos elementos gerenciados, como estações de trabalho, switches e roteadores. Cada elemento gerenciado possui uma MIB correspondente, que é padronizada e deve representá-lo de forma que qualquer ferramenta baseada em SNMP seja capaz de gerenciá-lo através da MIB correspondente.

O trabalho em [28] apresenta uma avaliação do SNMP como interface de gerência de redes virtuais utilizando a *Virtual Router Extended MIB* para gerenciar roteadores virtuais. Segundo [28], o SNMP foi considerado uma solução adequada para a gerência de roteadores virtuais, uma vez que oferece um método simplificado e uniforme para controlar e monitorar os dispositivos gerenciados. Em [29], afirma-se que a definição de uma MIB para a gerência de VM's seria um avanço no sentido do gerenciamento integrado de infraestruturas virtuais. Em [31] sugere-se que o SNMP seja utilizado como protocolo de monitoramento e controle em um sistema de gerência de VM's, pois um sistema de gerência assume que os elementos gerenciados oferecem uma interface de gerência, e o SNMP pode ser utilizado para definir tal interface.

O trabalho em [43] propõe uma MIB para o monitoramento de VM's em diferentes plataformas de virtualização, como VMWare, Xen, KVM e VirtualBox, devido à coexistência dessas plataformas nos centros de dados dos provedores de computação nas nuvens. No entanto, a MIB proposta em [43] não permite controlar as VM's e possui apenas um pequeno conjunto de objetos de gerência, deixando de incluir dados importantes sobre as VM's. Mais detalhes sobre o trabalho em [43] serão apresentados no capítulo 4. Os fabricantes de VMM's, como a VMWare e a Citrix, incluem em seus produtos interfaces de monitoramento em SNMP, que serão apresentadas com maior profundidade também no capítulo 4. Outros trabalhos recentes também fazem uso do *framework* SNMP, como o trabalho em [48], que utiliza o SNMP em um sistema de monitoramento de infraestrutura de GRID, e o trabalho em [23], que utiliza o SNMP para gerência de uma arquitetura distribuída de roteadores baseados em software.

Este trabalho propõe o uso do SNMP para monitoramento e controle de máquinas virtuais. Para definir uma interface padronizada para a gerência de máquinas virtuais, apresenta a *Virtual-Machines-MIB*, uma MIB que permite gerenciar diferentes VMM's através do protocolo SNMP. A *Virtual-Machines-MIB* reside na máquina física e, diferente das MIB's existentes para o VMWare e para o Xen, permite ao gerente não apenas monitorar aspectos da máquina física e das VM's, mas também executar ações de controle, como criar, excluir, reiniciar, ligar, desligar e congelar VM's, entre outras. Também é possível alterar o nome, a quantidade de memória RAM e de CPU's das VM's, além de alterar as unidades de armazenamento das mesmas.

Resultados práticos são apresentados utilizando ferramentas de gerência SNMP comuns para gerenciar diferentes monitores de máquinas virtuais. Para isso, foram criados agentes SNMP que oferecem suporte à *Virtual-Machines-MIB* e instalados em máquinas com o KVM e o Xen. Os agentes foram criados com base no agente SNMP de domínio público NET-SNMP [15], que foi estendido utilizando as funções da *libvirt* [9] para que passe a oferecer suporte à *Virtual-Machines-MIB*. A *libvirt* é uma interface de programação de aplicações para gerência de diversos monitores de máquinas virtuais.

A estrutura desse trabalho está organizado da seguinte forma: o capítulo 2 apresenta o

Simple Network Management Protocol (SNMP), o capítulo 3 apresenta as tecnologias de virtualização, o capítulo 4 descreve a gerência de máquinas virtuais, e o capítulo 5 descreve a *Virtual-Machines-MIB*, sua arquitetura de implementação e os resultados práticos. Por fim, a conclusão é apresentada no capítulo 6.

CAPÍTULO 2

O *SIMPLE NETWORK MANAGEMENT PROTOCOL*

Uma grande rede de computadores não pode ser gerenciada somente por meio de esforço humano [51]. Para que a gerência seja eficaz é necessário um sistema automatizado de gerência, que deve permitir ao gerente controlar e monitorar os diversos dispositivos que compõem a rede.

O *Simple Network Management Protocol* (SNMP) [25] é a arquitetura de gerência de redes mais amplamente utilizada. Um sistema de gerência SNMP é composto por estações de gerência, componentes gerenciados, informações de gerência e um protocolo de gerência. A arquitetura clássica de gerência de redes é mostrada na Figura 2.1. Nela, a estação de gerência comunica-se com os componentes gerenciados através de um protocolo de gerência. Os componentes gerenciados obtém as informações de gerência em sua base de dados interna.

A arquitetura de gerência define uma base de informações de gerência ou *management information base* (MIB), que contém um conjunto de objetos de gerência através dos quais as aplicações de gerência podem monitorar e controlar entidades gerenciadas [36]. O monitoramento das entidades gerenciadas consiste em ler periodicamente o valor de determinados objetos de gerência. O controle dessas entidades consiste em alterar o valor de determinados objetos de gerência, muitas vezes disparando uma determinada ação na entidade gerenciada.

A estação de gerência de rede, ou *network management station* (NMS), é um dispositivo que serve de interface entre o gerente de redes humano e o sistema de gerenciamento de redes. Deve conter um conjunto de aplicativos para apresentação e análise de dados, controle e monitoramento de dispositivos de rede e uma base de dados para armazenar as informações extraídas das MIB das entidades gerenciadas. Atualmente existe um grande número de sistemas de gerência de redes que utilizam o SNMP, tanto de código fechado

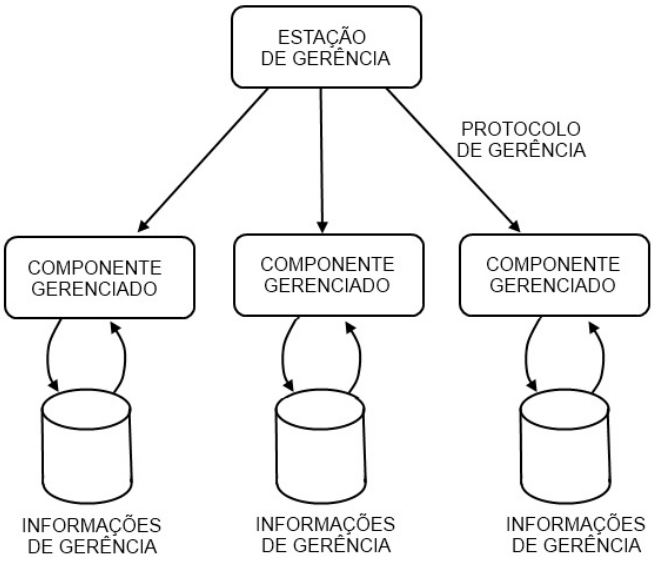


Figura 2.1: A arquitetura clássica de gerência do SNMP.

quanto aberto. Entre os sistemas de código aberto podem ser citados o MRTG [16], o Cacti [3], o Cricket [5] e o Zabbix [6]. Como exemplos de sistemas de código fechado podem ser citados o *Cisco Fabric Manager* [4], o *IBM Tivoli* [8] e o *HP Network Node Manager* [7].

Cada componente gerenciado possui um agente SNMP, que responde a pedidos de informações provenientes da estação de gerência e executa ações por ela comandadas. A estação de gerência e o agente SNMP comunicam-se através do protocolo de gerência SNMP. As informações que a estação de gerência pode solicitar ao agente correspondem aos objetos da MIB. Tanto a estação de gerência quanto o agente devem conhecer a MIB, para que a estação de gerencia saiba quais objetos podem ser lidos ou alterados no agente.

Os objetos na MIB que permitem somente operações de leitura (*read-only*) são os objetos de monitoramento, pois permitem que a estação de gerência monitore aspectos do componente gerenciado através da leitura periódica do valor desses objetos. Os objetos que permitem que a estação de gerência altere seus valores (*read-write*) são os objetos de controle, pois a alteração dos valores desses objetos causa alterações no comportamento do componente gerenciado. Além das operações de leitura e escrita originadas pela estação de gerência, também existe uma operação de notificação. Essa operação, denominada

TRAP, origina-se no agente SNMP com o objetivo de notificar a estação de gerência caso ocorram determinados eventos.

A arquitetura de gerência do SNMP define um relacionamento de "um-para-muitos" entre uma estação de gerência e um conjunto de nós gerenciados. A estação de gerência pode ler e alterar objetos nos nós gerenciados, e receber comunicados dos eventos que estão acontecendo nos mesmos. Este relacionamento entre uma estação de gerência e diversos nós gerenciados é implementado através das comunidades SNMP.

O SNMP define na RFC 1157 [25] o conceito de comunidade SNMP. A comunidade SNMP permite que o nó gerenciado possa limitar tanto o acesso às MIB's somente para gerentes autorizados (autenticação dos gerentes) quanto o nível de privilégio do gerente (controle de acesso). Cada comunidade SNMP deve ter um nome único, e as estações de gerência devem informar o nome da comunidade em cada mensagem de leitura ou escrita. Do ponto de vista da autenticação, o nome da comunidade serve como uma senha, pois a mensagem é considerada autêntica se o remetente conhece a comunidade. Do ponto de vista do controle de acesso, um agente pode limitar o acesso à determinados objetos da MIB baseado no nome da comunidade que o gerente informar.

Além da versão inicial, existem mais duas versões do SNMP que adicionam novas características e corrigem problemas encontrados na primeira versão. A versão 2 do SNMP (SNMPv2) foi proposta em 1993 através das RFC 1441 [27] e 1452 [26], e a versão 3 do SNMP (SNMPv3) foi lançada em 1998 [36].

O SNMPv2 adicionou as mensagens GETBULK e InformRequest, que não existiam na primeira versão do SNMP, e atualizou as versões da SMI para SMIv2 e da MIB para MIB-II[35]. Os tipos de mensagens existentes no protocolo SNMP serão descritos na seção 2.2. O SNMPv2 também introduziu a possibilidade de utilizar uma arquitetura de gerência tanto centralizada quanto distribuída. Na arquitetura de gerência distribuída, alguns nós irão atuar tanto como agentes quanto como gerentes. Quando no papel de agente, um nó deve aceitar comandos de um nó superior de gerência. Quando no papel de gerente intermediário, deve obter informações dos agentes a ele subordinados e repassar ao gerente superior.

Devido à inexistência de mecanismos de segurança eficazes na primeira versão do SNMP, como privacidade e autenticação de mensagens, o SNMPv2 foi originalmente proposto contendo uma série de mecanismos de segurança. Porém, em 1996, as especificações foram revisadas e os aspectos do SNMPv2 ligados à segurança das informações foram retirados, permanecendo somente o mecanismo de nomes de comunidades provenientes da primeira versão do SNMP. Por isso, essa versão revisada do SNMPv2 é conhecida como "SNMPv2 baseado em comunidade" ou SNMPv2C. O SNMPv2C é a versão do protocolo mais amplamente utilizada até o presente momento [47].

Para corrigir as deficiências de segurança do SNMPv1/v2, a versão 3 do SNMP (SNMPv3) foi lançada em 1998 [36] como um conjunto de padrões. Esse conjunto de documentos não traz uma especificação completa do SNMP, apenas define uma arquitetura geral para o SNMP e acrescenta mecanismos de segurança. Esses mecanismos de segurança podem ser usados com o SNMPv1 ou SNMPv2, e proporcionam autenticação das mensagens, privacidade e controle de acesso [52].

2.1 Base de informações de gerência

A MIB define os objetos que representam os diversos aspectos dos componentes gerenciados. Um componente possui diversos objetos, cada um armazenando uma determinada informação sobre esse componente. Esse conjunto de objetos varia entre os diversos equipamentos que podem ser encontrados nas redes, como computadores, switches, roteadores e impressoras. Por exemplo, a MIB de um roteador deverá conter objetos como número de octetos enviados e recebidos, interfaces de rede, endereços IP de redes de origem e endereços IP de destino. A MIB de um roteador é diferente da MIB de uma impressora, que irá conter dados como número de páginas impressas, uso do toner, dados de contato e outros.

A MIB é uma coleção de objetos estruturada em forma de árvore. A Figura 2.2 mostra a organização hierárquica de uma MIB padrão. Os objetos folha na árvore são os objetos realmente gerenciados. Todos os sistemas em uma rede de computadores que forem gerenciados por SNMP (estações de trabalho, servidores, roteadores, switches) devem

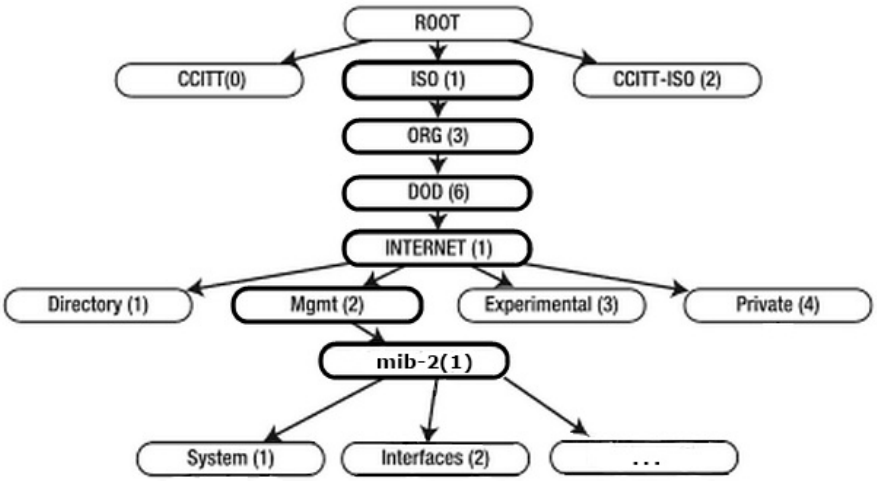


Figura 2.2: Organização hierárquica dos objetos de uma MIB conforme a SMI.

obedecer a um formato comum para a estruturação dos dados na árvore. Esse formato é definido pela Estrutura da Informação de Gerência, ou SMI (*Structure of Management Information*), especificada na RFC 1155 [45]. A SMI define a estrutura da árvore e os tipos de dados que podem ser usados na MIB, e especifica como os objetos da MIB são representados e nomeados.

A ASN.1 (*Abstract Syntax Notation One*) é uma linguagem formal para criação de sintaxes abstratas de dados, utilizada para definir uma MIB. Esta linguagem permite que sejam definidos tipos de objetos primitivos que podem ser combinados para obter objetos mais complexos. Cada tipo de objetos tem uma identificação e uma sintaxe de estrutura de dados e codificação. A sintaxe de codificação define como os tipos ASN.1 são codificados para a transmissão. Essa sintaxe evita ambiguidades, permitindo aos diversos componentes enviarem informações sem problemas com a representação dos dados.

A Figura 2.2 mostra os níveis hierárquicos da MIB conforme definidos na SMI. O primeiro nível após a raiz tem três nós: *iso* (1), *ccitt* (0) e *ccitt-iso* (2). Abaixo do nó *iso* encontra-se a subárvore *org* (3) para uso por outras organizações, sendo uma delas o Departamento de Defesa dos Estados Unidos, ou *dod* (6). Uma subárvore abaixo de *dod* foi alocada para administração da Internet, chamada *internet* (1). Após o nó *internet* encontram-se quatro nós: *directory* (1), *mgmt* (2), *experimental* (3) e *private* (4). Abaixo

do nó *mgmt*, encontra-se a *mib-2* (RFC 1213) [46], que é a MIB padrão encontrada nos dispositivos de rede atuais.

Abaixo do nó *private* existe somente uma subárvore, chamada *enterprises*, utilizada pelos fabricantes que se registram para então receber um identificador nessa subárvore e incluir informações de gerência de seus produtos. A subárvore *experimental* também pode ser utilizada por novas aplicações, que posteriormente podem vir a ser movidas para a subárvore *mgmt*.

Cada objeto em uma MIB possui um identificador ASN.1 do tipo *Object Identifier* (OID). Um OID consiste em uma lista de inteiros, separados por pontos, correspondendo aos nós percorridos da raiz até um objeto folha. Por exemplo o OID "1.3.6.1.2.1" está destacado na Figura 2.2. Existe também uma forma legível de representação de um OID, que consiste em uma série de nomes separados por pontos, cada um representando um nó da árvore. Por exemplo, o OID ".iso.org.dod.internet.mgmt.mib" corresponde a "1.3.6.1.2.1". O identificador ".iso.org.dod.internet.mgmt.mib" é usado como prefixo para outros grupos de objetos, cada qual com um identificador próprio, como o grupo *system*, que possui o identificador ".iso.org.dod.internet.mgmt.mib.system". Dentro dos grupos ocorrem ramificações que levam até os objetos de gerência, tais como o objeto *sysDescr*, que é um dos objetos do grupo *system* identificado como ".iso.org.dod.internet.mgmt.mib.system.sysDescr".

Segundo a linguagem ASN.1, as variáveis de uma MIB podem ser somente tipos de dados simples e tabelas. As variáveis simples incluem tipos como inteiros, inteiros sem sinal e strings de caracteres. As tabelas correspondem a vetores de variáveis com uma dimensão. Como regra de diferenciação entre tipos de dados simples e dados tabulares, se ao final do identificador o número for 0 (zero), trata-se de uma variável simples. Caso contrário, trata-se de uma tabela, onde o número corresponde ao campo que identifica unicamente cada entrada na tabela.

2.2 O protocolo SNMP

A estação de gerência e os agentes comunicam-se através do protocolo SNMP. O SNMP é um protocolo da camada de aplicação. Cada mensagem é codificada através da sin-

taxe ASN.1 e transmitida pela camada de transporte através em um único datagrama UDP (*User Datagram Protocol*). O SNMP utiliza a porta UDP 161 para enviar e receber requisições, e a porta 162 para receber *traps*. O SNMP implementa 6 mensagens principais: 3 mensagens para a estação de gerência obter informações (GET, GETNEXT, GETBULK), uma mensagem para a estação de gerência alterar configurações (SET) e duas mensagens utilizadas pelos agentes (RESPONSE e TRAP).

A mensagem GET permite que a estação de gerência obtenha o valor de um determinado objeto do agente. A mensagem GETNEXT retorna ao gerente o próximo objeto da MIB depois do objeto solicitado. Essa mensagem serve para percorrer uma árvore desconhecida de objetos. GETBULK é um mecanismo para obtenção de diversos objetos de uma só vez. A mensagem SET permite ao gerente alterar o valor dos objetos no agente. A mensagem RESPONSE é enviada pelo agente em resposta a uma consulta do gerente. Uma mensagem TRAP é enviada caso o agente tenha que notificar a estação de gerência sobre um determinado evento. Existe também a mensagem INFORMREQUEST, que serve para um gerente notificar outro gerente sobre algum evento significante, similar ao mecanismo utilizado pela TRAP.

O capítulo a seguir aborda os sistemas de máquinas virtuais, os métodos utilizados por tais sistemas e os desafios encontrados na virtualização.

CAPÍTULO 3

SISTEMAS DE MÁQUINAS VIRTUAIS

Este capítulo apresenta as principais técnicas utilizadas pelos sistemas de virtualização. Primeiramente é apresentada a virtualização completa, em seguida a paravirtualização, logo após a emulação e finalmente é abordada a virtualização em nível de sistema operacional.

A virtualização é uma tecnologia que permite que mais de uma instância de sistema operacional seja executada em uma mesma máquina física ao mesmo tempo. Uma camada de virtualização oferece suporte em baixo nível para a criação de múltiplas máquinas virtuais (*Virtual Machines* ou VM's), que são independentes e isoladas umas das outras, onde serão instalados os respectivos sistemas operacionais. Essa camada de virtualização é chamada de monitor de máquinas virtuais, ou *virtual machines monitor* (VMM) [41]. O sistema operacional da máquina virtual é chamado de sistema operacional convidado, e o da máquina física é chamado de sistema operacional hospedeiro.

Os monitores de máquinas virtuais permitem executar um grande número de máquinas virtuais em uma mesma máquina física ao mesmo tempo, permitindo assim a consolidação de diversos servidores em uma única máquina física [41]. O VMM deve prover o isolamento entre as máquinas virtuais. O isolamento compreende isolamento de falhas e isolamento de performance. O isolamento de falhas é responsável por impedir que uma falha em uma aplicação em determinada VM interfira nas demais VM's. O isolamento de performance é responsável por garantir que os requisitos de performance de uma determinada VM, baseados em acordos em nível de serviços (*Service Level Agreements* ou SLA's), serão cumpridos [32].

Diversos sistemas que implementam a virtualização foram desenvolvidos utilizando diferentes técnicas para atingir o objetivo de oferecer máquinas virtuais independentes e isoladas umas das outras. Cada técnica implica em diferentes consequências práticas,

ou seja, pode oferecer um nível maior ou menor de isolamento entre as VM's, oferecer melhor desempenho em determinados casos, suportar uma maior variedade de sistemas operacionais convidados ou permitir o uso de diferentes arquiteturas de CPU nas VM's.

3.1 Técnicas de virtualização

As principais técnicas adotadas no desenvolvimento de sistemas de máquinas virtuais são a virtualização completa, a paravirtualização, a emulação e a virtualização em nível de sistema operacional. A virtualização completa é feita replicando-se em software a arquitetura de hardware, de modo que o sistema operacional convidado possa ser executado sobre o software exatamente como se estivesse no hardware verdadeiro. A paravirtualização exige que o sistema operacional convidado seja modificado através da substituição das instruções privilegiadas da CPU por instruções que invocam o VMM. A emulação baseia-se na inserção de uma camada de software que realiza a interpretação de cada instrução de uma arquitetura de origem para uma arquitetura de destino. A virtualização em nível de sistema operacional consiste em fazer com que o sistema operacional hospedeiro forneça isolamento entre vários ambientes operacionais distintos.

3.1.1 Virtualização completa

A virtualização completa é feita replicando-se em software a arquitetura de hardware, de modo que o sistema operacional convidado possa ser executado sobre o software exatamente como se estivesse no hardware verdadeiro. Exemplos de sistemas que utilizam essa técnica são o VMWare [20] e o Microsoft Hyper-V [11].

Para que o VMM possa oferecer ao sistema operacional convidado uma interface equivalente ao hardware sem perder o controle sobre a máquina real, ele deve controlar e arbitrar as tentativas de acesso ao hardware. As tentativas do convidado de executar instruções privilegiadas no processador devem ser interceptadas pelo VMM, que assume o controle da máquina para decidir se irá executar a instrução ou não, ou se irá simular sua execução. As instruções não privilegiadas não precisam ser interceptadas pelo VMM,

podendo ser executadas diretamente no hardware - a chamada execução direta.

A arquitetura `IA-32` (x86) é a arquitetura de computadores mais amplamente utilizada. O conjunto de instruções da `IA-32` implementa um mecanismo de níveis de privilégio ou anéis de execução, que restringe a execução de determinadas instruções do processador somente aos níveis com maiores privilégios. Existem os níveis de privilégios de 0 a 3, sendo 0 para mais privilegiado e 3 para menos privilegiado (Figura 3.1). Esses níveis determinam se instruções privilegiadas podem ser executadas sem gerar exceção. Quando uma exceção for disparada, o processador irá escalonar para execução o software em nível 0, que deverá tratar tal exceção.

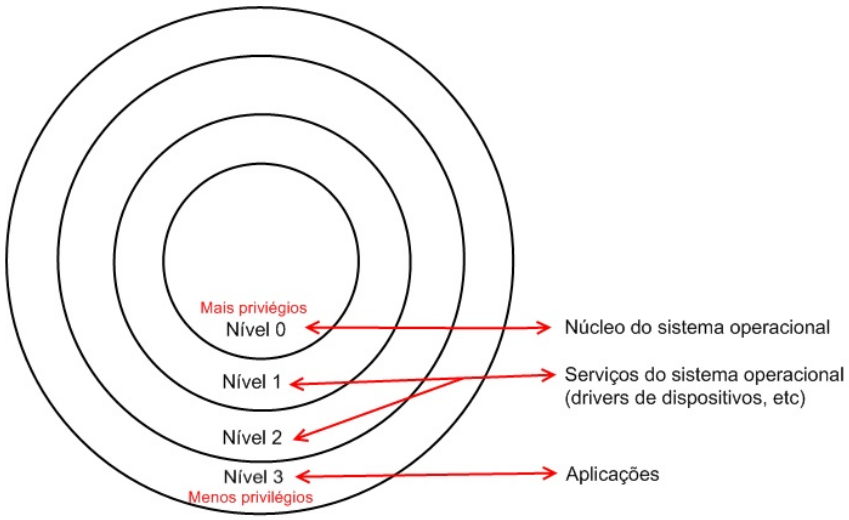


Figura 3.1: Anéis de execução Intel X86.

Normalmente, o sistema operacional é executado em nível 0, para que tenha controle sobre a CPU, e as aplicações são executadas em nível 3. Quando houver um VMM, adota-se um modelo em que o VMM é executado em nível 0, o sistema operacional convidado é executado em nível 1 e os demais programas em nível 3, o que dá ao VMM controle total sobre o processador e permite aos convidados privilégios maiores do que os programas executados neles (Figura 3.2). Assim, quando um convidado tentar executar uma instrução privilegiada, o processador irá disparar uma exceção e passar o controle para o VMM.

O conjunto de instruções da arquitetura `IA-32` pode ser dividido em três grupos:

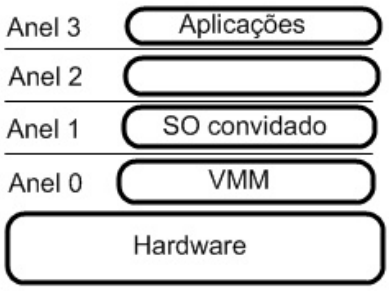


Figura 3.2: Distribuição dos níveis de privilégios na arquitetura X86.

instruções privilegiadas, instruções sensíveis de controle e instruções sensíveis de comportamento [44]. Segundo [44], um VMM pode ser construído para determinada arquitetura se o conjunto de instruções sensíveis for um subconjunto de conjunto de instruções privilegiadas, ou seja, as instruções sensíveis também devem gerar exceções se forem executadas sem o nível de privilégios correto.

A arquitetura `IA-32` não foi projetada para ser virtualizada, uma vez que possui um pequeno conjunto de instruções sensíveis que não fazem parte do conjunto de instruções privilegiadas. Quando o sistema operacional convidado tenta executá-las sem privilégios suficientes, elas não disparam uma exceção, dificultando sua interceptação pelo VMM. Esse é um dos desafios encontrados para a criação de um VMM sobre a arquitetura `IA-32`. Outro desafio é que algumas instruções não privilegiadas fornecem informações sobre a operação da CPU, como o nível de privilégio atual. Assim, é possível que o sistema operacional convidado execute alguma dessas instruções e determine que não está executando em modo privilegiado do processador.

Nos últimos anos, os fabricantes de hardware desenvolveram tecnologias que auxiliam a virtualização do processador, facilitando o desenvolvimento de VMM's e melhorando seu desempenho. O dois principais fabricantes de microprocessadores com suporte à arquitetura `IA-32` fornecem produtos com suporte à virtualização. A *Intel* com a tecnologia *Vanderpool* e a *AMD* com a tecnologia *Pacifica* têm em comum a inclusão no processador de um modo de execução destinado ao VMM e outro modo de execução destinado às máquinas virtuais. Cada um desses modos de execução possui todos os níveis de privilégio encontrados em um processador `IA-32` comum. Assim, o VMM pode deixar as

máquinas virtuais executarem diretamente sobre o hardware em nível 0 no modo virtual. O processador irá automaticamente trocar para o modo de operação destinado ao VMM quando um convidado tentar executar instruções privilegiadas, não mais sendo necessário que o VMM se encarregue de interceptar essas instruções. Isso simplificou os VMM's e fez com que surgissem soluções de virtualização como o KVM, exclusiva para processadores com esse recurso.

3.1.2 Paravirtualização

Na paravirtualização, a máquina virtual executa um sistema operacional modificado, em que as instruções privilegiadas foram substituídas por outras mais convenientes. Assim, diferente da virtualização completa, não existe a necessidade de interceptar qualquer instrução que venha a ser executada pelos convidados. Dessa maneira, o sistema operacional convidado é executado em uma interface diferente do hardware, criada no VMM. O inconveniente dessa abordagem é que o sistema operacional convidado deve ser modificado, o que pode não ser possível no caso de sistemas proprietários como o Microsoft Windows. Essa técnica é utilizada por sistemas como o Xen [22] e o Denali [53].

Os sistemas de máquinas virtuais que antes restringiam-se somente à paravirtualização passaram a aproveitar-se das tecnologias *Vanderpool* e *Pacifica* para fornecer mais possibilidades de virtualização [33]. Por exemplo, o Xen, que tradicionalmente empregava somente a paravirtualização e estava restrito a convidados previamente modificados, graças ao uso das funções de virtualização oferecidas pelo processador, passou a suportar também máquinas virtuais com sistemas operacionais não modificados.

3.1.3 Emulação

A emulação é uma técnica de virtualização baseada na interpretação de cada instrução de uma arquitetura de origem para uma arquitetura de destino. Por exemplo, um emulador da arquitetura x86 para um processador SPARC pode executar qualquer aplicação x86 em um processador SPARC, criando a ilusão que este é um processador x86. Para que isso aconteça, o emulador deve ser capaz de interpretar o conjunto de intruções do convidado

para o conjunto de intruções do hospedeiro. Exemplos de emuladores são o QEMU [13], o Bochs [2], o BIRD [1] e o Crusoe [37].

Comparando-se a emulação com a virtualização completa, apesar de ambas oferecerem em software a arquitetura de hardware e assim poderem executar sistemas operacionais não modificados, a emulação deve traduzir todas as instruções passadas ao processador, enquanto a virtualização completa permite que instruções não privilegiadas sejam executadas diretamente no hardware, obtendo assim um desempenho superior.

3.1.4 Virtualização em nível de sistema operacional

A virtualização em nível de sistema operacional é uma técnica de virtualização que permite a um único sistema operacional fornecer a ilusão de que possui vários sistemas virtuais. Essa técnica funciona essencialmente implementando quatro tipos de isolamento entre os sistemas virtuais: isolamento de sistemas de arquivos, de processos, de rede e de privilégios de administrador.

O isolamento entre sistemas de arquivos faz com que cada máquina virtual possua seu próprio sistema de arquivos raiz que na verdade estão dentro da árvore de diretórios do hospedeiro. O isolamento entre processos baseia-se na criação de diferentes contextos para cada máquina virtual, cujos processos são executados dentro desses contextos. O contexto do hospedeiro é o contexto "0" e possui mais privilégios que os demais: este contexto pode enxergar e terminar outros processos em outros contextos. O isolamento de rede faz com que todo o tráfego enviado por uma determinada interface de rede pertencente a uma VM seja alterado de forma que tenha sua origem no endereço IP da respectiva VM. O isolamento de privilégios de administrador baseia-se no particionamento de tais privilégios em subconjuntos, de forma que o administrador das máquinas virtuais tenha menos privilégios do que o administrador do hospedeiro, mas ainda assim mais privilégios do que as demais contas de usuários das máquinas virtuais.

A virtualização em nível de sistema operacional não é uma camada de virtualização propriamente dita, logo não tem de lidar com questões típicas de VMM's como por exemplo a interceptação de instruções enviadas para o processador. Essa técnica apoia-se

apenas no isolamento para criar a ilusão de que mais de uma máquina está sendo executada, porém internamente não existe mais de um sistema operacional em execução para lidar com todas as tarefas de cada máquina virtual. Dessa forma, possui a limitação de não oferecer suporte à virtualização de sistemas operacionais distintos do hospedeiro. O principal exemplo de sistema que utiliza essa técnica é o Linux-VServer [39].

CAPÍTULO 4

A GERÊNCIA DE MÁQUINAS VIRTUAIS

Sistemas de gerência de máquinas virtuais são essenciais para utilizar com eficiência os recursos de hardware disponíveis. Diversos trabalhos abordam o tema da gerência de máquinas virtuais, principalmente devido à difusão da computação nas nuvens e ao crescimento da quantidade de VM's. Este capítulo apresenta trabalhos relacionados à gerência de máquinas virtuais, e está organizado da seguinte forma: primeiramente são apresentados trabalhos que propõem uma abordagem de gerência de máquinas virtuais, em seguida são apresentadas as abordagens de gerência existentes e como se relacionam com a gerência de máquinas virtuais, e finalmente é apresentada a gerência de máquinas virtuais baseada em SNMP.

Em [54] propõe-se que a gerência de máquinas virtuais seja feita através de agentes em software instalados tanto no VMM quanto nos sistemas operacionais convidados. Os agentes permitem realizar diversas tarefas nas máquinas virtuais. Com eles, é possível obter dados internos dos sistemas operacionais convidados como processos em execução, utilização de recursos, unidades de armazenamento e interfaces de rede. Também permitem realizar auto-configuração e auto-reparo, instalar novas aplicações e executar rotinas *batch*. A comunicação entre gerente e agentes acontece através de RPC (*Remote Procedure Call*) e o método apresenta desempenho superior a outros como a *libvirt* ou as ferramentas para VMWare. Os inconvenientes desse método são a necessidade de instalação de software nas máquinas virtuais, a inclusão de um novo *daemon* nas máquinas físicas e virtuais e a ausência de integração dos agentes com diferentes sistemas de gerência.

O trabalho em [34] propõe uma arquitetura para gerência de máquinas virtuais baseada em REST (*REpresentational State Transfer*). O trabalho sugere que o REST pode substituir tanto os protocolos de comunicação entre o gerente humano e a estação de gerência, quanto entre a estação de gerência e os componentes gerenciados. Segundo [34],

a adoção do REST como única interface para transferência dos dados de gerência reduz as dificuldades em compartilhar esses dados entre múltiplas aplicações, aumentando a interoperabilidade entre as diversas aplicações de gerência.

A Open Cloud Computing Interface (OCCI) [12] define um protocolo e API para gerenciamento de nuvens também baseado em REST. A OCCI foi originalmente desenvolvida para criar uma API de gerenciamento remoto para nuvens com o modelo de serviço IaaS, tendo evoluído também para outros modelos como PaaS e SaaS. O trabelho em [30] apresenta em maiores detalhes o uso de REST e outras abordagens de gerência.

A *libvirt* [24] é uma interface de programação para o desenvolvimento de aplicações de gerência de máquinas virtuais capaz de gerenciar vários VMM's. A interface de gerência definida pela *libvirt* abrange desde o monitoramento de desempenho até a alocação e liberação de recursos de hardware das VM's. A *libvirt* será utilizada neste trabalho para a implementação da *Virtual-Machines-MIB* por oferecer suporte a diversos VMM's e abranger a maior parte das funcionalidades definidas pela *Virtual-Machines-MIB*, portanto será apresentada com maior profundida na seção 4.2.

Entre as abordagens de gerência existentes, é possível destacar o *Web-Based Enterprise Management* (WBEM), por possuir um conjunto de definições relacionadas com a gerência de máquinas virtuais, e o SNMP, que foi utilizado em diversos trabalhos para gerenciar máquinas virtuais.

O WBEM, definido pela *Distributed Management Task Force* (DMTF) e apoiado por diversos fabricantes de hardware e software, consiste em um conjunto de tecnologias padronizadas de comunicação e gerência desenvolvido para unificar a gerência de ambientes de computação distribuída. Caracteriza-se por permitir a troca de dados de gerência entre tecnologias e plataformas distintas, pois os dados transportados seguem padrões de codificação e transporte e são definidos dentro de um modelo comum de informação (*Common Information Model* ou CIM).

O CIM é um modelo conceitual de informação para descrever a administração de uma determinada entidade, que não está ligado a alguma implementação em particular. Isso permite a troca de informações de gerência entre sistemas de gerência, tanto "agente para

gerente"ou "gerente de gerente ". A DMTF define em [19] um esquema CIM para representar e gerenciar máquinas virtuais, demonstrando que é desejável incluir uma interface padronizada para o gerenciamento de máquinas virtuais também em outros *frameworks* de gerência. Uma comparação do WBEM, SNMP e outros *frameworks* pode ser encontrada em [30].

O *Simple Network Management Protocol* (SNMP), já apresentado anteriormente no Capítulo 2, também é utilizado para gerenciar máquinas virtuais. A seção a seguir apresenta trabalhos relacionados com a gerência de máquinas virtuais baseada em SNMP.

4.1 Gerência de máquinas virtuais baseada em SNMP

O SNMP é a arquitetura de gerência de redes mais amplamente difundida. Diversas ferramentas utilizam o SNMP como protocolo de gerência, aproveitando-se do conjunto existente de MIB's padronizadas, que são bastante difundidas, permitindo assim que as ferramentas de gerência possam gerenciar uma grande variedade de dispositivos.

Em [29], afirma-se que a definição de uma MIB para a gerência de VM's seria um avanço no sentido do gerenciamento integrado de infraestruturas virtuais. Em [31] sugere-se que o SNMP seja utilizado como protocolo de monitoramento e controle em um sistema de gerência de VM's, pois um sistema de gerência assume que os elementos gerenciados possuem uma interface de gerência, e o SNMP pode ser utilizado para definir tal interface.

O trabalho em [28] apresenta uma avaliação do SNMP como interface de gerência de redes virtuais, utilizando a *Virtual Router Extended MIB* para gerenciar roteadores virtuais. Segundo [28], o SNMP foi considerado uma solução adequada para a gerência de roteadores virtuais, uma vez que oferece um método rápido e uniforme para controlar e monitorar os dispositivos gerenciados. Outros trabalhos recentes também utilizam o *framework* SNMP. O trabalho em [48] utiliza o SNMP em um sistema de monitoramento de infraestrutura de GRID , e o trabalho em [23] utiliza o SNMP para gerência de uma arquitetura distribuída de roteadores baseados em software.

A *libvirt-snmp* [10] é um subprojeto da *libvirt* que fornece funcionalidade SNMP para a *libvirt*. Com a *libvirt-snmp*, é possível monitorar VM's, assim como configurar atributos

das VM's por SNMP. A *libvirt-snmp* permite ao gerente obter informações sobre as VM's, alterar o estado das VM's e ser informado sobre determinados eventos.

A *libvirt-snmp* apresenta a *LIBVIRT-MIB*, que define uma tabela de VM's onde cada linha contém o nome da VM, seu estado, número de CPUs, RAM, limite de RAM e tempo de CPU. A *LIBVIRT-MIB* permite alterar o estado de uma VM, os demais objetos são somente-leitura. O projeto *libvirt-snmp* apoia-se nas funcionalidades da *libvirt*, não tendo a intenção de incluir operações além daquelas oferecidas pela *libvirt*.

O trabalho em [43] salienta a importância de uma interface padronizada para o monitoramento de máquinas virtuais em várias plataformas de virtualização, como VMWare, Xen, KVM e VirtualBox, devido à coexistência dessas plataformas nos centros de dados de provedores de computação nas nuvens. Sugere que tal interface padronizada seja construída com base em SNMP, utilizando a *MIB-II* e a *Host Resources MIB* para monitorar a máquina física e propondo uma nova MIB para monitorar as VM's, chamada *NCNU-VM-MIB*.

Também é apresentada em [43] uma implementação da *NCNU-VM-MIB* em que a *libvirt* é utilizada para obter as informações sobre as VM's, realizando assim o monitoramento de máquinas virtuais em três VMM's distintos: VMWare, KVM e Xen. A *NCNU-VM-MIB* permite somente o monitoramento das VM's, não sendo possível controlar as mesmas através de SNMP. Devido ao pequeno número de objetos presentes na *NCNU-VM-MIB*, o trabalho não permite obter diversas informações sobre as máquinas virtuais, como o tipo de sistema operacional convidado e as unidades de armazenamento.

Alguns monitores de máquinas virtuais incluem interfaces de gerência em SNMP, como o VMWare ESX Server e o Citrix Xen Server. Um estudo sobre as MIB's utilizadas por esses VMM's oferece um panorama sobre diversas variáveis de interesse para a gerência de máquinas virtuais. As seções 4.1.1 e 4.1.2 tratam respectivamente das MIB's existentes para monitoramento do VMWare ESX Server e do Xen.

4.1.1 A MIB do VMWare ESX Server

O VMWare ESX Server inclui uma série de definições de MIB que formam a subárvore *vmware*, com o OID ".iso.org.dod.internet.private.enterprises.vmware" (1.3.6.1.4.1.6876). A MIB do VMWare pode ser visualizada na Figura 4.1.

A MIB do VMWare separa os dados sobre a máquina física dos dados sobre as máquinas virtuais colocando-os em grupos distintos. Os dados sobre a máquina física ficam no grupo *vmwSystem* que contém objetos que guardam o nome (*vmwProdName*) e a versão (*vmwProdVersion* e *vmwProdBuild*) do VMWare, e no grupo *vmwResources*, que contém objetos relacionados às CPUs (*vmwCPU*), memória (*vmwMemory*) e dispositivos de armazenamento (*vmwStorage*).

Os dados sobre as máquinas virtuais ficam no grupo *vmwVirtMachines*, onde fica a tabela *vmwVmTable*, que contém a lista de máquinas virtuais. Essa tabela contém dados sobre cada uma das VM's, como o nome (*vmwVmDisplayName*), o caminho para o arquivo de configuração da VM (*vmwVmConfigFile*), o sistema operacional em execução na VM (*vmwVmGuestOS*), a quantidade de memória disponível para a VM (*vmwVmMemSize*), o estado da VM (*vmwVmState*), o estado do sistema operacional da VM (*vmwVmGuestState*) e o número de CPU's virtuais alocadas para a VM (*vmwVmCpus*).

As demais tabelas do grupo *vmwVirtMachines* contém mais informações sobre cada uma das VM's, como informações sobre os *Host Bus Adapters* (HBA's), discos virtuais, interfaces de rede virtuais, drives de disquetes e de CD-ROM. Cada uma das entradas dessas tabelas faz referência às VM's da tabela *vmwVmTable* através do campo identificador da VM.

O subgrupo *vmwESX*(1) no grupo *vmwProductSpecific*(4) contém as *traps* SNMP. Existem *traps* que informam que uma VM foi ligada, desligada, que o sistema operacional da VM não respondeu ou voltou a responder, que uma máquina virtual foi suspensa e que foi detectada uma alteração no hardware físico.

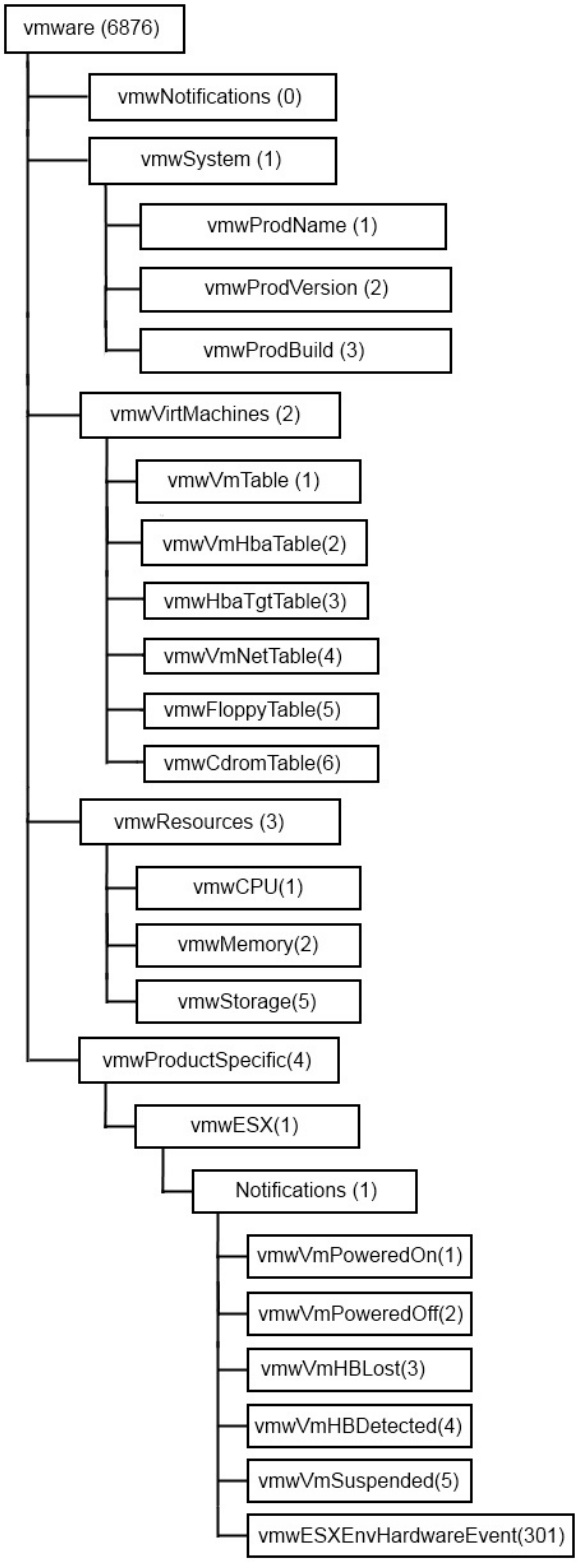


Figura 4.1: A MIB do VMWare.

4.1.2 A MIB do Xen

Um grupo de pesquisa da Universidade de Braunschweig, Alemanha, desenvolveu uma MIB para gerenciar máquinas virtuais Xen [17]. A *xenMIB* possui objetos para monitorar tanto o VMM quanto as máquinas virtuais em uma instalação do Xen. A Figura 4.2 mostra a *xenMIB*.

A *xenMIB* possui grupos de objetos distintos para os dados sobre a máquina física (*xenHost*) e sobre as máquinas virtuais (*xenDomainTable*), que para o Xen são chamadas "domínios virtuais". O subgrupo *xenHost* possui objetos contendo a versão do Xen (*xenHostXenVersion*), a quantidade total de memória disponível na máquina física (*xenHostTotalMemKBytes*), a quantidade de CPU's existentes na máquina física (*xenHostCPUs*) e a frequência das CPU's (*xenHostCPUMHz*).

A *xenDomainTable* contém o nome de cada domínio Xen (*xenDomainName*), seu estado (*xenDomainState*), a quantidade de memória em uso (*xenDomainMemKBytes*) e a quantidade total de memória (*xenDomainMaxMemKBytes*).

As tabelas *xenVCPUTable*(3) e *xenNetworkTable*(4) contém dados sobre as CPU's virtuais e sobre as redes de cada domínio virtual.

4.1.3 Discussão sobre as MIB's apresentadas

Tanto a MIB do VMWare quanto do Xen separam os dados sobre a máquina física dos dados sobre as máquinas virtuais colocando-os em grupos de objetos distintos. Ambas as MIB's possuem uma tabela com a lista de máquinas virtuais existentes e seus principais atributos, como nome, estado e quantidade de memória. Outros dados sobre as máquinas virtuais, como informações sobre CPU's, HBA's, discos virtuais, interfaces de rede virtuais, drives de disquetes e de CD-ROM, podem ser encontrados em outras tabelas, em que cada entrada está relacionada a uma VM através de um campo identificador da VM.

As MIB's apresentadas para gerência do VMWare ESX e do Xen apresentam apenas objetos para o monitoramento dos recursos das máquinas físicas e virtuais. Desse modo, não é possível alterar o valor dos objetos de gerência, o que seria necessário para realizar o controle das máquinas virtuais. Além disso, as MIB's apresentadas não seguem um

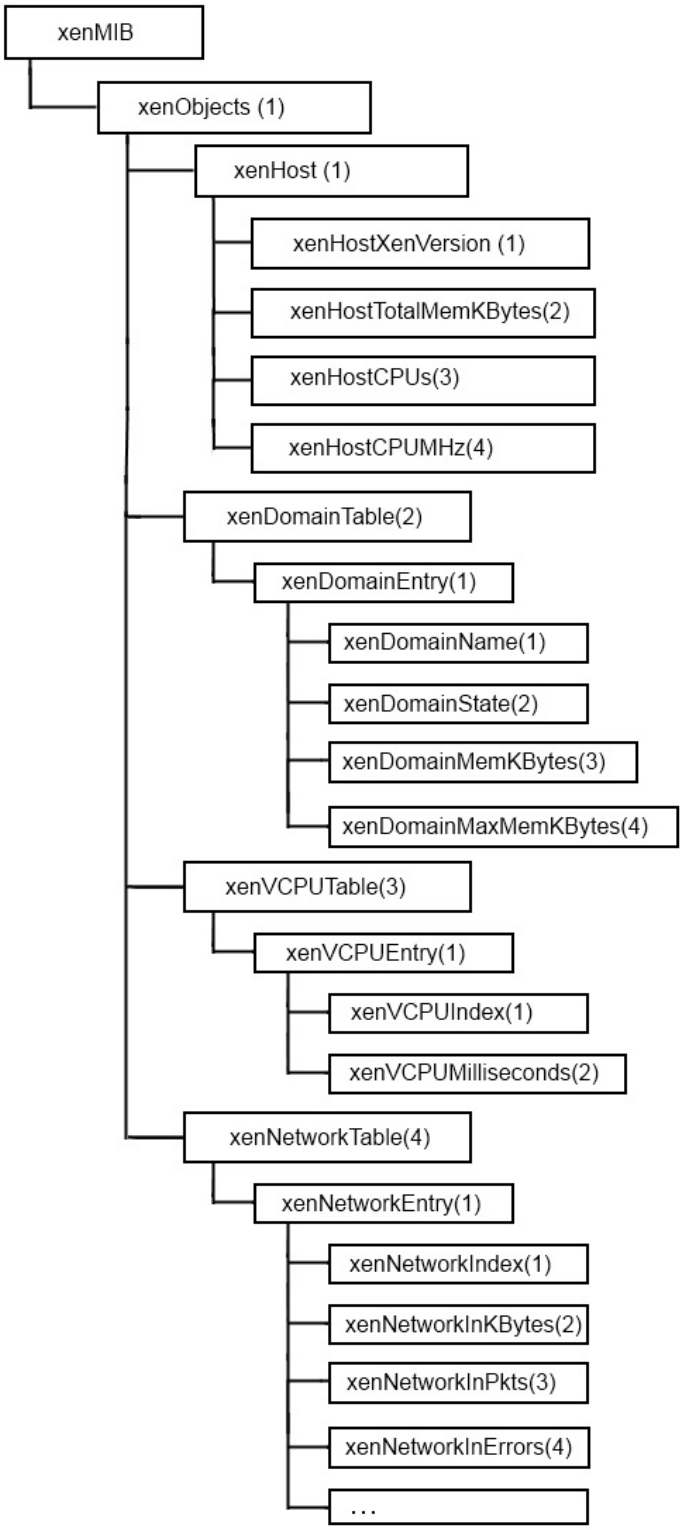


Figura 4.2: A MIB do Xen e seus principais objetos de monitoramento.

padrão, o que tem como consequência a impossibilidade de que uma aplicação desenvolvida para monitorar o VMWare através de SNMP possa monitorar também o Xen. A Tabela 4.1 mostra uma comparação entre os objetos encontrados nas MIB's do VMWare e do Xen.

4.2 A *libvirt*

A *libvirt* [24] é uma biblioteca de programação para o desenvolvimento de aplicações capazes de gerenciar diversos monitores de máquinas virtuais. Atualmente a libvirt oferece suporte a diversos monitores de máquinas virtuais, como Xen, KVM, QEMU, Virtual Box e VMWare. Ela introduz um *middleware* que interage com o VMM para oferecer uma interface padronizada para o desenvolvimento de aplicações de gerência de VM's.

A *libvirt* divide-se em duas partes: uma parte independente do VMM, e outra específica para cada VMM. Esta última é composta por *drivers*, um para cada VMM. Desse modo, para cada monitor de máquinas virtuais gerenciado, a *libvirt* deve possuir o *driver* correspondente. As aplicações utilizam a API pública da *libvirt*, que internamente faz o mapeamento para o *driver* adequado.

O objetivo da *libvirt* é prover uma camada comum e estável o suficiente para gerenciar máquinas virtuais com segurança e, se possível, remotamente [9]. A *libvirt* contém funções de gerência como provisionamento (instalação do sistema operacional convidado), criação, modificação, monitoramento, controle, migração e parada de máquinas virtuais. Oferece também funções para enumerar, monitorar e utilizar os recursos disponíveis na máquina física, como CPU's, memória, armazenamento e rede.

VMM's como o VMWare e o Hyper-V oferecem funções de gerência remota através dos protocolos SOAP e WS-Management, respectivamente. A libvirt também oferece uma interface para a gerência remota de máquinas virtuais. Essa interface é constituída pelo *driver* remoto, que fica na estação de gerência e comunica-se com um processo servidor (um *daemon*) no VMM gerenciado. Esse *daemon* chama-se *libvirtd*. A Figura 4.3 mostra de um lado uma estação de gerência executando uma ferramenta de gerência desenvolvida através da *libvirt* que está realizando a gerência de um servidor KVM através do *driver*

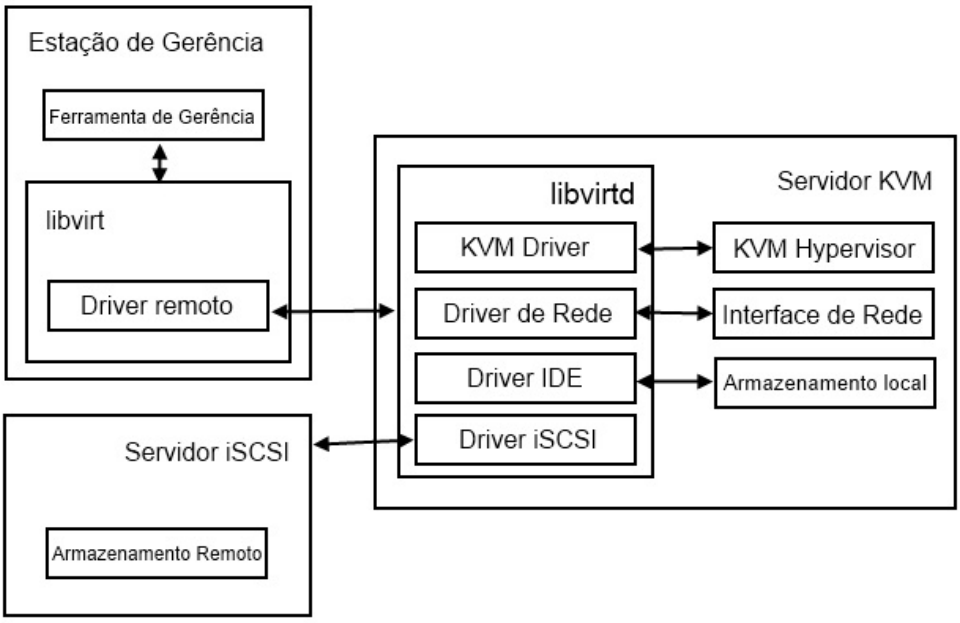


Figura 4.3: Arquitetura da libvirt baseada em drivers. Fonte: [24].

remoto.

Na arquitetura descrita na Figura 4.3, o servidor KVM deve conter tanto o *daemon libvirtd* quanto os *drivers* para gerência do KVM e outros. As requisições da estação de gerência são tuneladas pelo *driver* remoto para o *daemon libvirtd* no servidor KVM. O *daemon libvirtd* recebe as requisições e localmente invoca o *driver* adequado, que pode ser o *driver* KVM, que comunica-se com o monitor de máquinas virtuais do KVM (KVM *hypervisor*), o *driver* de rede, que comunica-se com a interface de rede existente no servidor KVM, o *driver* IDE, que controla o armazenamento local, e o *driver* iSCSI, que comunica-se com um servidor iSCSI remoto.

A *libvirt* é capaz de gerenciar remotamente tanto VMM's que executam o *daemon libvirtd* quanto o VMWare e o Hyper-V, que não são capazes de executar esse *daemon*. Nesses casos, a *libvirt* utiliza diretamente as interfaces de gerência remota oferecidas por esses VMM's, respectivamente em SOAP e WS-Management. Atualmente, a *libvirt* é utilizada por diversos aplicativos de gerência de máquinas virtuais, como o Virsh [9], o *Virtual Machines Manager* [18], o *oVirt* [50] e o OpenNebula [49].

Tabela 4.1: Comparação entre os objetos encontrados nas MIB's do VMWare e do Xen. Campos marcados com × indicam que o objeto não consta na MIB.

Descrição	**VMWare MIB**	**Xen MIB**
Nome do VMM	vmwProdName	×
Versão do VMM	vmwProdVersion	xenHostXenVersion
Compilação do VMM	vmwProdBuild	×
Tabela de máquinas virtuais	vmwVmTable	xenDomainTable
Índice da VM na tabela	vmwVmIdx	xenDomainName
Nome da VM	vmwVmDisplayName	xenDomainName
Arquivo de configuração da VM	vmwVmConfigFile	×
Sistema operacional convidado	vmwVmGuestOS	×
Memória em megabytes da VM	vmwVmMemSize	xenDomainMaxMemKBytes
Memória em uso pela VM	×	xenDomainMemKBytes
Estado da VM	vmwVmState	xenDomainState
Estado do sistema operacional da VM	vmwVmGuestState	×
Número de CPU's virtuais da VM	vmwVmCpus	xenVCPUTable
Milisegundos utilizados pela CPU virtual	×	xenVCPUMilliseconds
Informações sobre os HBA's das VM's	vmwVmHbaTable	×
Discos virtuais configurados nas VM's	vmwHbaTgtTable	×
Interfaces de rede das VM's	vmwVmNetTable	xenNetworkTable
Drives de disquetes para cada VM	vmwFloppyTable	×
Drives de CD-ROM para cada VM	vmwCdromTable	×
Número de CPU's físicas	vmwNumCPUs	xenHostCPUs
Frequência das CPU's físicas	×	xenHostCPUMHz
Quantidade de memória física	mwMemSize	xenHostTotalMemKBytes
Memória física disponível	vmwMemAvail	×
Número de HBA's	vmwHostBusAdapterNumber	×
Nome do HBA	vmwHbaDeviceName	×
Estado do HBA	vmwHbaStatus	×
Modelo do HBA	vmwHbaModelName	×
Driver do HBA	vmwHbaDriverName	×
PCI Id do HBA	vmwHbaPci	×
Notificações ou alertas	Notifications	xenTraps

CAPÍTULO 5

A *VIRTUAL-MACHINES-MIB*

Atualmente existem MIB's que permitem monitorar determinados VMM's, como as MIB's do VMWare e do Xen, e outras, como a *NCNU-VM-MIB* e a *LIBVIRT-MIB* que, apesar de não serem direcionadas a um VMM em particular, não contemplam determinadas funcionalidades necessárias para a gerência de máquinas virtuais, como alterar o estado ou as configurações de hardware da VM. Outro problema é que, diferente das MIB's utilizadas no gerenciamento de redes, que são padronizadas e amplamente difundidas, tais MIB's apresentam conjuntos de informações distintos, não obedecendo a um padrão.

O conjunto de objetos da *Virtual-Machines-MIB* inclui diversos objetos das demais MIB's relacionadas à virtualização e adiciona novos objetos a esse conjunto com o objetivo de controlar o estado das VM's e alterar as configurações de hardware virtual.

A *Virtual-Machines-MIB* permite obter dados do VMM, como nome e versão, listar as VM's e obter dados de cada uma delas, como nome, informações sobre as CPU's, RAM, discos, sistema operacional convidado, interfaces de rede e HBA's. Permite também criar e excluir VM's, controlar o estado das VM's e alterar as configurações de hardware virtual (memória RAM, CPU's e discos virtuais). Para criar novas VM's ou conectar novos dispositivos de armazenamento nas VM's, a *Virtual-Machines-MIB* define modelos (ou *templates*), tornando essas operações mais simples e diretas.

Diferente da maior parte das MIB's existentes, a *Virtual-Machines-MIB* inclui não somente objetos de *read-only*, mas também objetos *read-write*, permitindo, além do monitoramento, também o controle das máquinas virtuais. A Tabela 5.1 mostra os objetos de monitoramento encontrados na *Virtual-Machines-MIB* e se os mesmos estão presentes nas MIB's do VMWare e do Xen. A Tabela 5.2 mostra os objetos de controle encontrados na *Virtual-Machines-MIB*. Diversos dos objetos de controle encontrados na *Virtual-Machines-MIB* também estão presentes nas demais MIB's, no entanto apenas

como objetos de monitoramento.

Tabela 5.1: Objetos de monitoramento encontrados na *Virtual-Machines-MIB* e se os mesmos estão presentes (•) ou ausentes (×) das MIB's do VMWare e do Xen.

Descrição	**Virtual-Machines-MIB**	**VMWare MIB**	**XenMIB**
Nome do VMM	vmmName	•	×
Versão do VMM	vmmVersion	•	•
Compilação do VMM	×	•	×
Identificador único da VM	vmUUID	•	×
Nome do SO convidado	osName	•	×
Versão do SO convidado	osVersion	×	×
Fabricante do SO convidado	osVendor	×	×
Kernel do convidado	kernelName	×	×
Parâmetros do kernel do convidado	kernelParams	×	×
Arquivo de configuração da VM	vmConfFile	•	×
Memória em uso pela VM	memoryUsedKB	×	•
Memória livre na VM	memoryFreeKB	×	×
Arquitetura da CPU virtual	cpuArch	×	×
Frequência da CPU virtual	cpuMhz	×	×
Milissegundos utilizados pela CPU virtual	cpuMilliseconds	×	•
Drives de disquetes da VM	storageTable	•	×
Drives de CD-ROM da VM	storageTable	•	×
Informações sobre os HBA's das VM's	hbaTable	•	×
Interfaces de rede das VM's	networkTable	•	•

Este capítulo está organizado da seguinte forma: primeiramente será apresentada a organização da *Virtual-Machines-MIB*, em seguida a arquitetura de implementação da *Virtual-Machines-MIB*, e finalmente são apresentados os resultados experimentais.

5.1 Organização da *Virtual-Machines-MIB*

A *Virtual-Machines-MIB* é organizada em uma subárvore no grupo *VirtualMachines*. Esse grupo, definido pela *Virtual-Machines-MIB*, contém, no primeiro subnível, nove tabelas e dois subgrupos, que são mostrados na Figura 5.1(a). A descrição completa em ASN.1 da *Virtual-Machines-MIB* pode ser encontrada no apêndice deste trabalho. A descrição em ASN.1 da *Virtual-Machines-MIB* foi verificada sintática e semanticamente com base

Tabela 5.2: Objetos de controle encontrados na *Virtual-Machines-MIB* e se os mesmos estão presentes (•) ou ausentes (×) das MIB's do VMWare e do Xen como objetos de monitoramento.

Descrição	**Virtual-Machines-MIB**	**VMWare MIB**	**XenMIB**
Lista de máquinas virtuais	virtualMachinesTable	•	•
Nome da VM	vmName	•	•
Memória em megabytes da VM	memoryTotalMb	•	•
Estado da VM	vmState	•	•
Número de CPU's virtuais da VM	cpuTable	•	•
Discos virtuais das VM's	storageTable	•	×
Modelo (*template*) para criação de VM's	virtualMachineTemplateTable	×	×

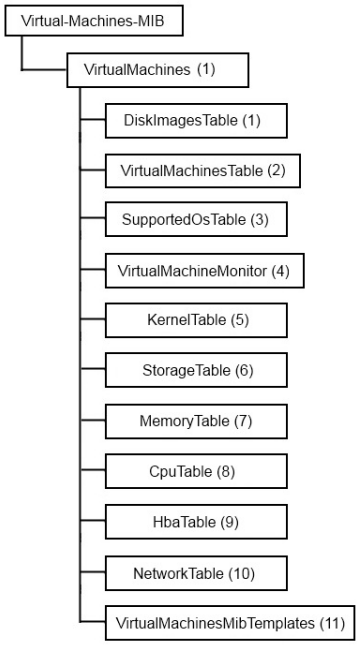
na SMIv2 através da ferramenta *smilint* [21].

As VM's estão listadas na tabela *VirtualMachinesTable* (Figura 5.1(b)), que contém informações sobre cada VM. O objeto *VmUUID* contém o UUID [38] de cada VM. O UUID foi escolhido como identificador das VM's para que seja possível identificar unicamente uma determinada VM até mesmo ao gerenciar diversas máquinas físicas com uma grande quantidade de VM's.

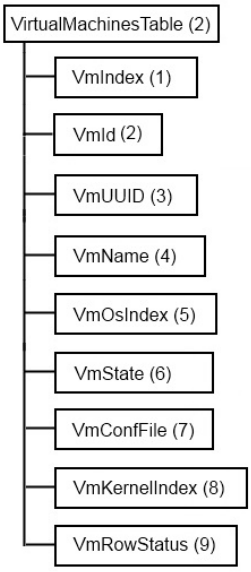
O objeto *VmState* permite ler e alterar o estado da VM, com base nos estados sugeridos pela *Distributed Management Task Force* (DMTF) em [14] e [19]. Através desse objeto é possível definir, ativar, desativar, congelar, suspender, desligar e reiniciar uma máquina virtual.

O objeto *VmName* permite ler e alterar o nome da VM. *VmOsIndex* aponta para uma entrada em *SupportedOsTable* (Figura 5.2(a)), que contém informações sobre o sistema operacional convidado. *VmConfFile* informa o nome do arquivo de configuração da VM, que pode ser um arquivo com a descrição em XML do domínio virtual [9] ou um arquivo texto com formato específico para cada VMM. *VmKernelIndex* aponta para uma entrada em *KernelTable* (Figura 5.2(b)), que contém informações sobre o *kernel* de sistema operacional utilizado na VM. *VmRowStatus* contém o estado de cada linha conceitual da tabela. Este objeto é responsável por criar e excluir VM's.

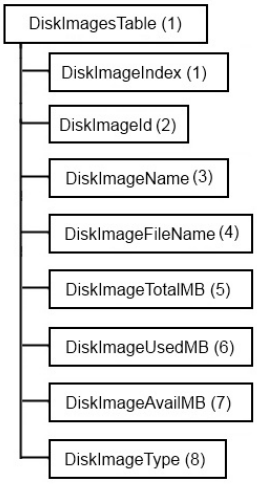
A tabela *DiskImagesTable* contém os arquivos de imagens de discos que podem ser



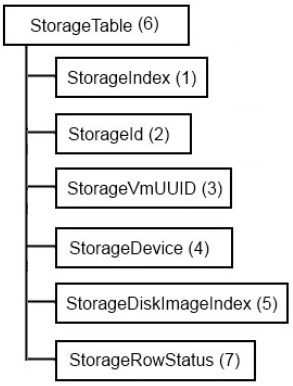
(a) O grupo *VirtualMachines*.



(b) A tabela *VirtualMachinesTable*

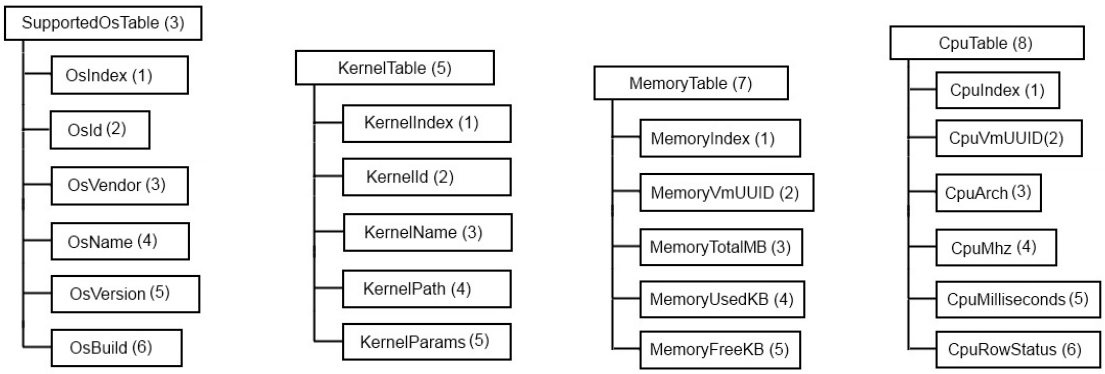


(c) A tabela *DiskImagesTable*.



(d) A tabela *StorageTable*.

Figura 5.1: Virtual-Machines-MIB: O grupo *VirtualMachines* e as tabelas *VirtualMachinesTable*, *DiskImagesTable* e *StorageTable*.



(a) A tabela *SupportedOsTable* (b) A tabela *KernelTable* (c) A tabela *MemoryTable*. (d) A tabela *CpuTable*.

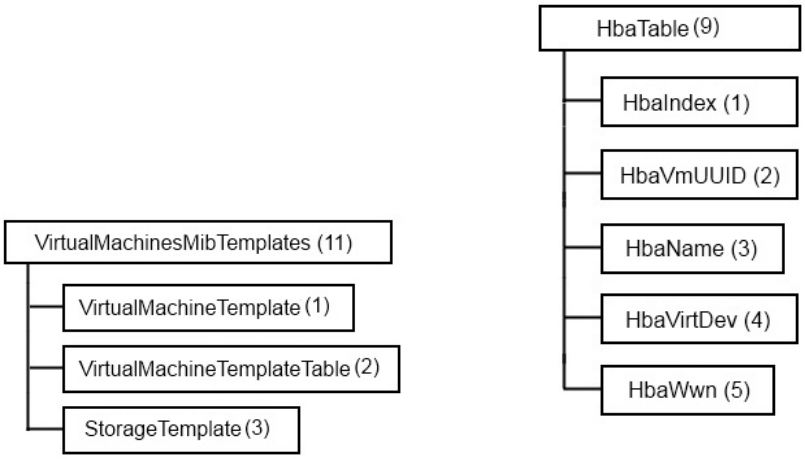
Figura 5.2: Virtual-Machines-MIB: As tabelas *SupportedOsTable*, *KernelTable*, *MemoryTable* e *CpuTable*.

utilizados como dispositivos de armazenamento. Os objetos da tabela *DiskImagesTable* podem ser visualizados na Figura 5.1(c).

A tabela *StorageTable* contém as imagens de discos atualmente alocadas às VM's como dispositivos de armazenamento. Cada entrada nesta tabela constitui um relacionamento entre uma VM e uma imagem de disco, uma vez que as imagens de disco são a base dos dispositivos de armazenamento virtuais. Essa tabela permite conectar e desconectar as imagens de disco das VM's, assim como substituir uma imagem já conectada a uma VM. Os objetos de *StorageTable* podem ser visualizados na Figura 5.1(d).

As tabelas *MemoryTable* e *CpuTable* contém informações sobre sobre o uso de memória RAM e das CPU's em cada uma das VM's. A Figura 5.2(c) mostra os objetos de *MemoryTable*, e a Figura 5.2(d) mostra os objetos de *CpuTable*. Os objetos *memoryVmUUID* e *cpuVmUUID* relacionam cada uma das entradas dessas tabelas com uma VM da tabela *VirtualMachinesTable* através do seu UUID.

O grupo *VirtualMachineMonitor* contém informações sobre o VMM local, como nome (*VMMName*) e versão (*VMMVersion*). A tabela *HbaTable* contém informações sobre os *Host Bus Adapters* virtualizados conectados às VM's. A tabela *NetworkTable* contém informações sobre as conexões de rede de cada uma das VM's, como endereçamento IP, MAC, *throughput* da interface e outras. As Figuras 5.3(b) e 5.4(a) contém respectivamente



(a) O grupo *VirtualMachinesMibTemplates*.

(b) A tabela *HbaTable*.

Figura 5.3: Virtual-Machines-MIB: Objetos das tabelas *HbaTable* e *NetworkTable*

os objetos das tabelas *HbaTable* e *NetworkTable*.

A *Virtual-Machines-MIB* utiliza modelos (*templates*) para as operações de criação de VM's e de conexão de um disco virtual em uma VM. Os modelos são armazenados no grupo *virtualMachinesMibTemplates*, cuja organização e uso serão explicados na seção a seguir.

5.1.1 Modelos de criação de VM's e dispositivos de armazenamento

As operações de criação de VM's e de conexão de um disco virtual em uma VM são realizadas incluindo novas linhas em determinadas tabelas. Ao incluir uma nova linha em uma tabela, todos os campos da nova linha devem ser preenchidos com algum valor. A inclusão de uma nova linha é realizada através de uma mensagem que faz o agente SNMP incluir mais uma linha na tabela. Como não é possível informar os valores de cada um dos campos nesta mensagem, os campos recebem inicialmente os valores armazenados em um modelo ou *template*.

A operação de criar uma VM é realizada alterando-se o valor do campo *vmRowStatus* de uma entrada qualquer de *virtualMachinesTable* para o inteiro "5", correspondente à

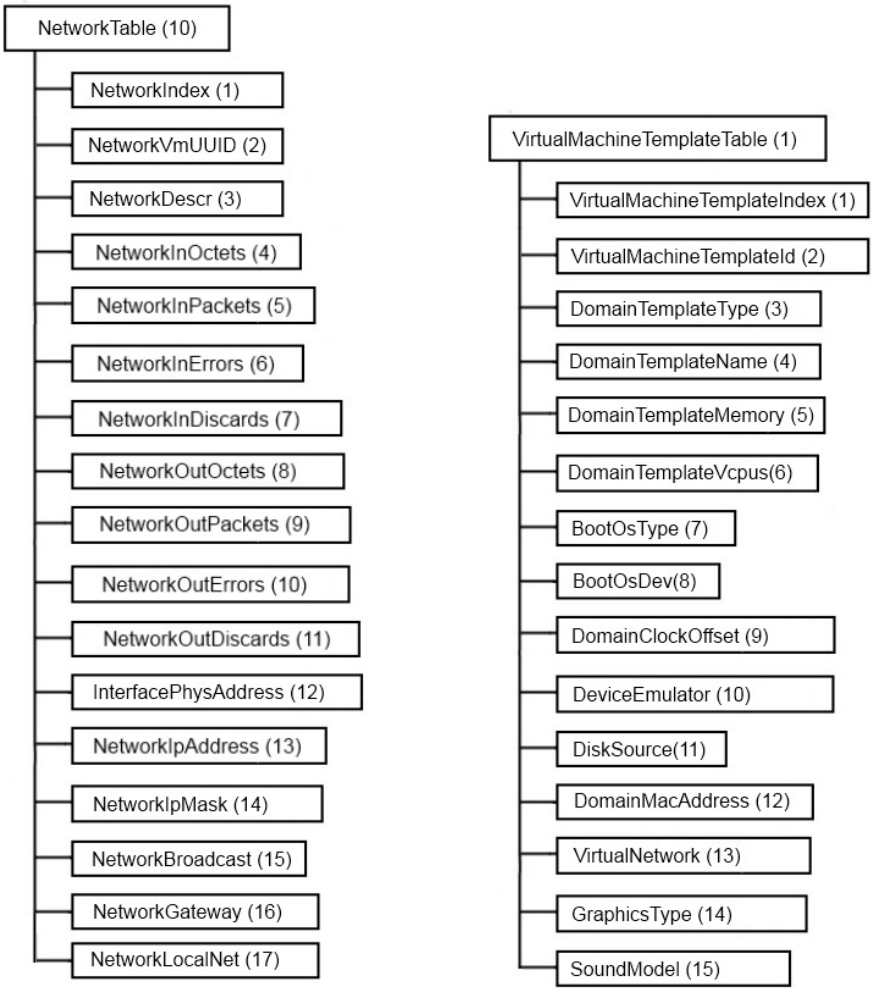
operação *createAndWait* [40]. A operação de conectar um disco virtual em uma VM consiste em alterar o valor do campo *storageRowStatus* da tabela *storageTable* na linha correspondente à VM para o inteiro "4", correspondente à operação *createAndGo*. Essas operações incluem mais uma linha nas respectivas tabelas, no entanto não informam os valores para todos os campos da nova linha.

Assim, uma vez disparada a criação de uma nova VM, uma linha é incluída na tabela *virtualMachinesTable* e campos como nome da VM, quantidade de memória, quantidade de CPU's e outros são obtidos do modelo. Da mesma forma, ao disparar a operação de conexão de um disco virtual em uma VM, uma linha é incluída na tabela *storageTable* e o nome do disco virtual a ser conectado à VM é obtido do modelo.

A *Virtual-Machines-MIB* armazena os modelos no grupo *virtualMachinesMibTemplates* (Figura 5.3(a)). A tabela *virtualMachineTemplateTable* armazena os modelos para a criação de novas VM's, o objeto *virtualMachineTemplate* armazena qual o modelo de criação de VM atualmente em uso, e o objeto *storageTemplate* armazena a imagem de disco a ser utilizada ao conectar um disco virtual em uma VM.

A tabela *virtualMachineTemplateTable* permite armazenar diversos modelos, e o objeto *virtualMachineTemplate* indica qual entrada de *virtualMachineTemplateTable* corresponde ao modelo atualmente em uso. Também é possível criar novos modelos e alterar os modelos existentes. Uma vez que a nova VM foi criada com base no modelo atual, a *Virtual-Machines-MIB* permite que o gerente altere as configurações da VM recém-criada, alterando seu nome, quantidade de memória, CPU's e discos virtuais. Os objetos da tabela *virtualMachineTemplateTable* podem ser visualizados na Figura 5.4(b).

O objeto *storageTemplate* armazena um inteiro que corresponde ao campo *diskImageIndex* da tabela *diskImagesTable*. Este campo identifica uma entrada em *diskImagesTable* que corresponde à imagem de disco modelo atualmente em uso, a qual é conectada às VMs quando um novo dispositivo de armazenamento é criado. Uma vez criada a nova linha em *storageTemplate* e consequentemente o novo dispositivo de armazenamento na VM, temporariamente conectado à imagem de disco modelo, o gerente pode alterar a linha recém-criada para que aponte para outra imagem de disco.



(a) A tabela *NetworkTable*.

(b) A tabela *VirtualMachineTemplate-Table*.

Figura 5.4: Virtual-Machines-MIB: O grupo *virtualMachinesMibTemplates* e a tabela *VirtualMachineTemplateTable*.

5.2 Arquitetura de implementação da *Virtual-Machines-MIB*

A arquitetura de implementação da *Virtual-Machines-MIB* baseia-se em um agente SNMP, encarregado de obter as informações e realizar as operações definidas na MIB. A estação de gerência externa deve comunicar-se com o agente através do protocolo SNMP. O agente SNMP deve obter as informações que constituem a MIB a partir de diversas fontes, como as interfaces de programação do sistema operacional hospedeiro e do VMM, a *libvirt*, e os arquivos XML com os modelos de criação de VM's.

A arquitetura descrita pode ser visualizada na Figura 5.5. Na figura, é possível visualizar que a estação de gerência comunica-se com a agente SNMP utilizando os objetos definidos na *Virtual-Machines-MIB*. O agente SNMP obtém os dados e realiza as operações solicitadas pela estação de gerência através de diversos meios, como a API do sistema operacional hospedeiro (OS API), a API do VMM e a *libvirt*. O agente SNMP também lê e escreve os modelos de VM, armazenados em arquivos XML. Além disso, na Figura 5.5 é possível visualizar que a *libvirt* utiliza a API do VMM para gerenciá-lo, o que acontece através dos *drivers*, conforme descrito anteriormente na seção 4.2.

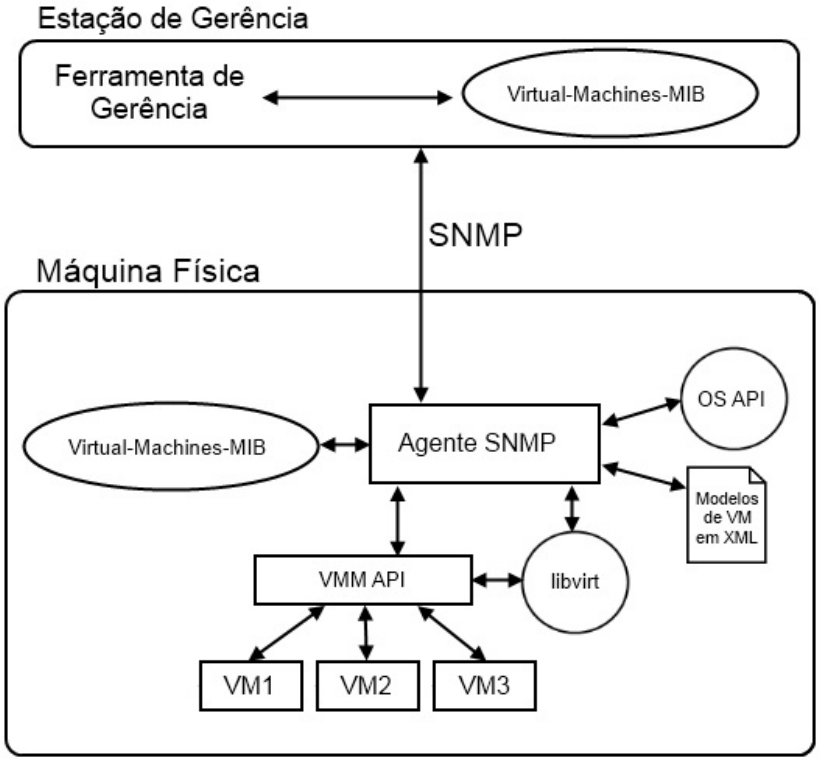


Figura 5.5: Arquitetura de implementação da *Virtual-Machines-MIB*.

A maior parte dos dados da *Virtual-Machines-MIB* podem ser obtidos através da *libvirt*, no entanto as tabelas *supportedOsTable*, que contém informações sobre os sistemas operacionais convidados, e *kernelTable*, que contém informações sobre o *kernel* dos sistemas operacionais convidados, não são informados pela *libvirt*, sendo necessário obtê-los a partir de outras fontes, como por exemplo a API do VMM.

As tabelas *NetworkTable* e *HbaTable* também possuem objetos que não podem ser obtidos através da *libvirt*, sendo necessário obtê-los de outras fontes que variam em função do VMM em uso, como a API do VMM ou do sistema operacional hospedeiro.

Determinadas funcionalidades da *Virtual-Machines-MIB* que alteram os recursos de uma VM, apesar de utilizarem funções da *libvirt*, a mesma não oferece a função equivalente. Nesses casos, deve-se utilizar a *libvirt* para obter a descrição em XML da VM, alterar os dados do XML e redefinir a VM a partir do XML alterado. A Lista 5.1 apresentada a descrição de uma VM em XML no formato definido pela *libvirt* [9].

Após alterar o XML que descreve a VM, a mesma deve ser redefinida com o XML alterado. A operação de redefinição de uma VM consiste em recriar a VM utilizando um XML com as novas configurações da VM mas com o mesmo UUID, através da função `virDomainDefineXML`. Nesses casos, a alteração realizada terá efeito somente no próximo boot da VM.

Por exemplo, o objeto *vmName*, da tabela *virtualMachinesTable*, permite alterar o nome de uma VM. Para implementar essa funcionalidade, é necessário obter a descrição da VM em XML através da função `virDomainGetXMLDesc`. Em seguida, o mesmo deve ser traduzido através de um *parser XML*, e o nome da VM deve ser alterado. Finalmente, a VM deve ser redefinida com o novo XML.

A *libvirt* também não oferece uma função para listar os discos virtuais existentes em uma VM, uma informação necessária para obter os dados da tabela *storageTable*. O método utilizado para obter essa informação consiste em obter a descrição em XML de cada VM através da função `virDomainGetXMLDesc` e utilizar um *parser XML* para obter os dispositivos de armazenamento a partir do XML.

Os modelos de criação de VM, armazenados na tabela *virtualMachineTemplateTable*,

Lista 5.1: Descrição de uma VM em XML.

```
<domain type='kvm'>
  <name>vm-debian-1</name>
  <uuid>a0216fad-8fed-15eb-4cb8-7f84b08b7bb9</uuid>
  <memory>131072</memory>
  <currentMemory>65536</currentMemory>
  <vcpu>1</vcpu>
  <os>
    <type arch='x86_64' machine='pc-0.12'>hvm</type>
    <boot dev='hd'/>
  </os>
  <features>
    <acpi/>
    <apic/>
    <pae/>
  </features>
  <clock offset='utc'/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <devices>
    <emulator>/usr/bin/kvm</emulator>
    <disk type='file' device='disk'>
      <driver name='qemu' type='raw'/>
      <source file='/var/lib/libvirt/images/vm-debian-1.img'/>
      <target dev='vda' bus='virtio'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0'/>
    </disk>
    <disk type='file' device='disk'>
      <driver name='qemu' type='raw'/>
      <source file='/var/lib/libvirt/images/template-vol.img'/>
      <target dev='vdb' bus='virtio'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x07' function='0x0'/>
    </disk>
    <disk type='block' device='cdrom'>
      <driver name='qemu' type='raw'/>
      <target dev='hdc' bus='ide'/>
      <readonly/>
      <address type='drive' controller='0' bus='1' unit='0'/>
    </disk>
    <controller type='ide' index='0'>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x1'/>
    </controller>
    <interface type='network'>
      <mac address='52:54:00:5c:ec:57'/>
      <source network='default'/>
      <model type='virtio'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
    </interface>
    <serial type='pty'>
      <target port='0'/>
    </serial>
    <console type='pty'>
      <target type='serial' port='0'/>
    </console>
    <input type='mouse' bus='ps2'/>
    <graphics type='vnc' port='-1' autoport='yes'/>
    <sound model='ac97'>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0'/>
    </sound>
    <video>
      <model type='cirrus' vram='9216' heads='1'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0'/>
    </video>
    <memballoon model='virtio'>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x0'/>
    </memballoon>
  </devices>
</domain>
```

são provenientes de arquivos XML armazenados no hospedeiro que descrevem um domínio virtual no formato definido pela *libvirt* [9]. O XML é traduzido pela *Virtual-Machines-MIB* e com base nele é inserido um modelo na *virtualMachineTemplateTable*, que pode ser alterado por SNMP através de operações de escrita da *Virtual-Machines-MIB*. A Lista 5.2 apresenta um arquivo XML utilizado para inserir um modelo na tabela *virtualMachineTemplateTable*.

Lista 5.2: Modelo de uma VM em XML.

```
<domain type="kvm">
  <name>new-vm-from-template</name>
  <uuid/>
  <memory>131072</memory>
  <vcpu>1</vcpu>
  <os>
    <type>hvm</type>
    <boot dev="hd"/>
  </os>
  <features>
    <acpi/>
    <apic/>
    <pae/>
  </features>
  <clock offset="utc"/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <devices>
    <emulator>/usr/bin/kvm</emulator>
    <disk type="file" device="disk">
      <source file="/var/lib/libvirt/images/template-vol.img"/>
      <target dev="vda" bus="virtio"/>
    </disk>
    <disk type="block" device="cdrom">
      <target dev="hdc" bus="ide"/>
      <readonly/>
    </disk>
    <controller type="ide" index="0">
    </controller>
    <interface type="network">
      <mac address="aa:bb:cc:dd:ee:ff"/>
      <source network="default"/>
    </interface>
    <input type="mouse" bus="ps2"/>
    <graphics type="vnc" port="-1" autoport="yes"/>
    <serial type="pty">
      <target port="0"/>
    </serial>
    <console type="pty">
      <target type="serial" port="0"/>
    </console>
    <sound model="ac97">
    </sound>
  </devices>
</domain>
```

A *Virtual-Machines-MIB* foi implementada neste trabalho com base no agente SNMP de domínio público *NET-SNMP* [15], que foi estendido em linguagem C para oferecer suporte aos objetos da *Virtual-Machines-MIB*. As extensões incluídas no agente SNMP *NET-SNMP* implementam a maior parte das funcionalidades oferecidas pela *Virtual-Machines-MIB*, como a obtenção de informações sobre o VMM e sobre as VM's, as trocas de estado das VM's e as alterações do hardware virtual das VM's. Cada uma das funcionalidades implementadas exige a invocação de diversas funções da API da *libvirt* a partir

do agente SNMP, como conectar-se com o VMM, enumerar as VM's e ler uma informação ou executar determinada ação sobre a VM desejada.

Cada uma das tabelas que compõem a *Virtual-Machines-MIB* foi implementada separadamente e compilada como um módulo do agente *NET-SNMP*, de acordo com as informações obtidas em [15]. *NetworkTable* e *HbaTable* não foram implementadas neste trabalho pois as informações contidas nas mesmas devem ser obtidas de fontes diferentes da *libvirt*, com isso aumentando o custo necessário para concluir a implementação e contribuindo pouco com o objetivo de validar a abordagem proposta pela *Virtual-Machines-MIB*.

5.3 Resultados experimentais

Foram obtidos resultados experimentais da *Virtual-Machines-MIB* instalando-se o agente SNMP estendidos em duas máquinas, uma com o KVM e outra com o Xen. O KVM foi instalado em uma estação de trabalho com CPU Intel Core i5-M480, com 2.67 GHz e 4 GB de RAM. O Xen foi instalado em uma estação de trabalho com CPU Intel Core i5-2520M, com 2.5 GHz e 4 GB de RAM. Ambas foram gerenciadas a partir da estação de gerência através de ferramentas tradicionais de gerência SNMP. A arquitetura utilizada nos experimentos pode ser visualizada na Figura 5.6.

Para demonstrar a utilização da *Virtual-Machines-MIB*, foram realizados os experimentos constantes na Tabela 5.3.

O primeiro experimento, que consiste em obter informações sobre as VM's, foi executado no KVM e no Xen. Os resultados podem ser visualizados na Tabela 5.4 para o KVM e na Tabela 5.5 para o Xen.

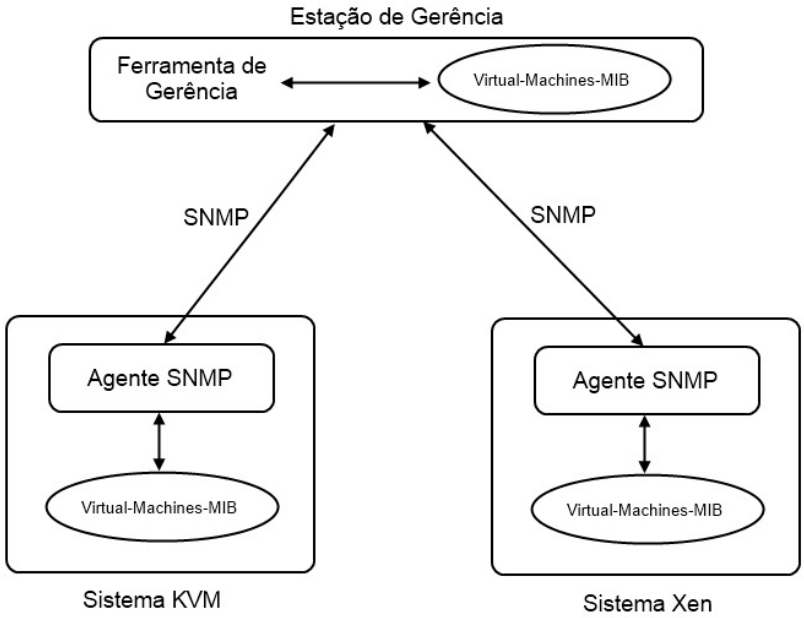


Figura 5.6: A arquitetura utilizada nos experimentos da *Virtual-Machines-MIB*.

Tabela 5.3: Experimentos realizados para demonstrar a utilização da *Virtual-Machines-MIB*.

Experimento	**Descrição**
1	Obter informações sobre as VM's
2	Criar uma VM
3	Alterar o nome da VM
4	Alterar a quantidade de memória RAM
5	Conectar um determinado disco virtual
6	Iniciar a VM
7	Inserir mais uma CPU virtual na VM
8	Excluir a VM

Tabela 5.4: Obtendo informações sobre as VM's no KVM.

virtualMachines

diskImagesTable

diskImageId	diskImageName	diskImageFileName	diskImageTotalMb	diskImageUsedMb	diskImage-AvailMb	diskImage-Type
1	vm-debian-1.img	/images/vm-debian-1.img	8192	897	7294	raw
2	template-vol.img	/images/template-vol.img	10	10	0	raw
3	vm-debian-2.img	/images/vm-debian-2.img	4096	0	4096	raw
4	vm-debian-3.img	/images/vm-debian-3.img	4096	0	4096	raw
5	vm-debian-4.img	/images/vm-debian-4.img	8192	0	8192	raw
6	vm-debian-5.img	/images/vm-debian-5.img	4096	0	4096	raw
7	vm-debian-6.img	/images/vm-debian-6.img	2048	0	2048	raw

virtualMachinesTable

vmId	vmUUID	vmName	vmOsIndex	vmState	vmConfFile	vmKernelIndex	vmRowStatus
1	7cae0a2c ...	vm-debian-4	1	running	/var/lib/libvirt/vm-debian-4.xml	1	active
2	a0216fad ...	vm-debian-1	1	running	/var/lib/libvirt/vm-debian-1.xml	1	active
3	05c6bfe1 ...	vm-debian-2	1	paused	/var/lib/libvirt/vm-debian-2.xml	1	notInService
4	945c6a0e ...	vm-debian-3	1	running	/var/lib/libvirt/vm-debian-3.xml	1	active
5	0be03b74 ...	vm-debian-6	1	shutoff	/var/lib/libvirt/vm-debian-6.xml	1	notInService
6	f4d32541 ...	vm-debian-5	1	running	/var/lib/libvirt/vm-debian-5.xml	1	notInService

supportedOsTable

osId	osVendor	osName	osVersion	osBuild
1	Debian	GNU/Linux	Lenny	2.6.32-5-amd64
2	Microsoft	Windows	Seven	64-bits

virtualMachineMonitor

- vmmName.0 = STRING: QEMU
- vmmVersion.0 = STRING: 12005

kernelTable

kernelId	kernelName	kernelPath	kernelParams
1	Debian Linux 2.6.32-5-amd64	/boot/vmlinuz-2.6.32-5-amd64	root=/dev/vda1 ro quiet

storageTable

storageId	storageVmUUID	storageDevice	storageDiskImageIndex	storageRowStatus
1	be03b74-5902-88ab-57b8-73c10caf0bd	disk	7	active
2	be03b74-5902-88ab-57b8-73c10caf0bd	cdRom	0	active
3	f4d32541-7618-1ed1-32-8d1a7bc2aaf2	disk	6	active
4	f4d32541-7618-1ed1-32-8d1a7bc2aaf2	cdRom	0	active
5	7cae0a2c-6d39-7ef4-8e77-e83ff68c7526	disk	5	active
6	7cae0a2c-6d39-7ef4-8e77-e83ff68c7526	cdRom	0	active
7	a0216fad-8fed-15eb-4cb8-7f84b08b7bb9	disk	1	active
8	a0216fad-8fed-15eb-4cb8-7f84b08b7bb9	disk	2	active
9	a0216fad-8fed-15eb-4cb8-7f84b08b7bb9	cdRom	0	active
10	5c6bfe1-cf32-7f6a-e84e-fd8ccca331aa	disk	3	active
11	5c6bfe1-cf32-7f6a-e84e-fd8ccca331aa	cdRom	0	active
12	945c6a0e-60bc-d029-dd7e-2e7d0a98a5c7	disk	4	active
13	945c6a0e-60bc-d029-dd7e-2e7d0a98a5c7	cdRom	0	active

memoryTable

memoryVmUUID	memoryTotalMb	memoryUsedKB	memoryFreeKB
be03b74-5902-88ab-57b8-73c10caf0bd	128	131072	0
f4d32541-7618-1ed1-32-8d1a7bc2aaf2	128	131072	0
7cae0a2c-6d39-7ef4-8e77-e83ff68c7526	128	131072	0
a0216fad-8fed-15eb-4cb8-7f84b08b7bb9	64	65536	0
5c6bfe1-cf32-7f6a-e84e-fd8ccca331aa	128	131072	0
945c6a0e-60bc-d029-dd7e-2e7d0a98a5c7	128	131072	0

cpuTable

cpuVmUUID	cpuArch	cpuMhz	cpuMilliseconds	cpuRowStatus
be03b74-5902-88ab-57b8-73c10caf0bd	x86_64	2661	0	active
f4d32541-7618-1ed1-32-8d1a7bc2aaf2	x86_64	2661	1665163520	active
7cae0a2c-6d39-7ef4-8e77-e83ff68c7526	x86_64	2661	935032704	active
a0216fad-8fed-15eb-4cb8-7f84b08b7bb9	x86_64	2661	3270065408	active
5c6bfe1-cf32-7f6a-e84e-fd8ccca331aa	x86_64	2661	55098112	active
945c6a0e-60bc-d029-dd7e-2e7d0a98a5c7	x86_64	2661	4015032704	active

virtualMachinesMibTemplates

- virtualMachineTemplate = 1
- storageTemplate = 2
- virtualMachineTemplateTable

Id	Type	Name	Memory	Vcpus	BootOsType	BootOsDev	ClockOffset	deviceEmu	disk
1	kvm	new-vm	131072	1	hvm	hd	utc	/usr/bin/kvm	template-vol.img

Tabela 5.5: Obtendo informações sobre as VM's no Xen.

virtualMachines

diskImagesTable

diskImageId	diskImageName	diskImageFileName	diskImageTotalMb	diskImageUsedMb	diskImageAvailMb	diskImageType
1	template-vol.img	/images/template-vol.img	10	0	10	raw
2	hvm-xen-1.img	/images/hvm-xen-1.img	10240	1049	9190	raw
3	vmxen-debian-1.img	/images/vmxen-debian-1.img	10240	980	9259	raw
4	vmxen-debian-n.img	/images/vmxen-debian-n.img	1000	0	1000	raw

virtualMachinesTable

vmId	vmUUID	vmName	vmOsIndex	vmState	vmConfFile	vmKernelIndex	vmRowStatus
1	0-0-0-0-0	Domain-0	0	running	/var/lib/libvirt/Domain-0.xml	0	active
2	8f915a94...	hvm-xen-1	0	running	/var/lib/libvirt/hvm-xen-1.xml	0	notReady
3	0e875600...	vmxen-debian-1	0	shutoff	/var/.../vmxen-debian-1.xml	0	notInService

supportedOsTable

osId	osVendor	osName	osVersion	osBuild
1	Debian	GNU/Linux	Lenny	2.6.32-5-amd64
2	Microsoft	Windows	Seven	64-bits

virtualMachineMonitor

- vmmName.0 = STRING: Xen
- vmmVersion.0 = STRING: 4000000

kernelTable

kernelId	kernelName	kernelPath	kernelParams
1	Debian Linux 2.6.32-5-amd64	/boot/vmlinuz-2.6.32-5-amd64	root=/dev/vda1 ro quiet

storageTable

storageId	storageVmUUID	storageDevice	storageDiskImageIndex	storageRowStatus
1	8f915a94-e61a-5d74-4e11-ee72a5d51971	disk	2	active
2	8f915a94-e61a-5d74-4e11-ee72a5d51971	cdRom	0	active
3	7818ca85-7359-ec02-7568-e488d9636af8	disk	4	active
4	7818ca85-7359-ec02-7568-e488d9636af8	cdRom	0	active
5	0e875600-4e82-43a2-4494-8ae188f81c8a	disk	3	active

memoryTable

memoryVmUUID	memoryTotalMb	memoryUsedKB	memoryFreeKB
0-0-0-0-0	2944	3014656	0
8f915a94-e61a-5d74-4e11-ee72a5d51971	027	1052636	0
7818ca85-7359-ec02-7568-e488d9636af8	128	131072	0
0e875600-4e82-43a2-4494-8ae188f81c8a	512	524288	0

cpuTable

cpuVmUUID	cpuArch	cpuMhz	cpuMilliseconds	cpuRowStatus
0-0-0-0-0	x86_64	2494	2957636705	active
0-0-0-0-0	x86_64	2494	2957636705	active
0-0-0-0-0	x86_64	2494	2957636705	active
0-0-0-0-0	x86_64	2494	2957636705	active
8f915a94-e61a-5d74-4e11-ee72a5d51971	x86_64	2494	393990947	active
7818ca85-7359-ec02-7568-e488d9636af8	x86_64	2494	3065496350	active
0e875600-4e82-43a2-4494-8ae188f81c8a	x86_64	2494	48852610	active

virtualMachinesMibTemplates

- virtualMachineTemplate = 1
- storageTemplate = 1

virtualMachineTemplateTable

Id	Type	Name	Memory	Vcpus	BootOsType	BootOsDev	ClockOffset	deviceEmu	disk
1	xen	new-vm	131072	1	hvm	hd	utc	/usr/lib/x...	template-vol.img

O segundo experimento consiste em criar uma nova VM. Para tal, o valor do campo *vmRowStatus* de *virtualMachinesTable* deve ser alterado para "5" (*createAndWait*) em qualquer linha da tabela. Com isso, uma nova VM é criada com base nas configurações do modelo em (*virtualMachineTemplateTable*).

Ao serem criadas, as novas VM's recebem com o nome definido no campo *domainTemplateName* da tabela *virtualMachineTemplateTable*. O terceiro experimento consiste em alterar o nome da VM, desse modo os nomes das VM's recém-criadas serão alterados para "vm-debian-1000" no KVM e "hvm-xen-1000"' no Xen. A Tabela 5.6 mostra o conteúdo de *virtualMachinesTable* no KVM e no Xen após os nomes terem sido alterados, com as

linhas correspondentes às novas VM's em destaque.

Tabela 5.6: Conteúdo de *virtualMachinesTable* no KVM e no Xen com as VM's recém-criadas.

Saída no KVM

vmId	vmUUID	vmName	vmOsIndex	vmState	vmConfFile	vmKernelIndex	vmRowStatus
1	7cae0a2c ...	vm-debian-4	1	running	/var/lib/libvirt/vm-debian-4.xml	1	active
2	a0216fad ...	vm-debian-1	1	running	/var/lib/libvirt/vm-debian-1.xml	1	active
3	05c6bfe1 ...	vm-debian-2	1	paused	/var/lib/libvirt/vm-debian-2.xml	1	notInService
4	945c6a0e ...	vm-debian-3	1	running	/var/lib/libvirt/vm-debian-3.xml	1	active
5	0be03b74 ...	vm-debian-6	1	shutoff	/var/lib/libvirt/vm-debian-6.xml	1	notInService
6	f4d32541 ...	vm-debian-5	1	running	/var/lib/libvirt/vm-debian-5.xml	1	notInService
7	**ff74ed37...**	**vm-debian-1000**	**1**	**shutoff**	**/var/lib/libvirt/vm-debian-1000.xml**	**1**	**notInService**

Saída no Xen

vmId	vmUUID	vmName	vmOsIndex	vmState	vmConfFile	vmKernelIndex	vmRowStatus
1	0-0-0-0-0	Domain-0	0	running	/var/lib/libvirt/Domain-0.xml	0	active
2	8f915a94 ...	hvm-xen-1	0	running	/var/lib/libvirt/hvm-xen-1.xml	0	notReady
3	e875600 ...	vmxen-debian-1	0	shutoff	/var/lib/libvirt/vmxen-d...	0	notInService
4	**738f9343...**	**hvm-xen-1000**	**0**	**shutoff**	**/var/lib/libvirt/hvm-xen-1000.xml**	**0**	**notInService**

Após trocar o nome da VM recém-criada, a quantidade de memória RAM deve ser alterada, conforme o experimento número 4 da Tabela 5.3. O conteúdo de *memoryTable* é apresentado na Tabela 5.7, com as linhas correspondentes às VM's recém-criadas em destaque. Verifica-se no campo *memoryTotalMb* da tabela que as VM's recém-criadas estão com 128 MB de RAM. Para alterar a quantidade de memória RAM de uma VM, este valor deve ser alterado para a quantidade de memória desejada, em *megabytes*.

Tabela 5.7: Conteúdo de *MemoryTable* no KVM e no Xen.

Saída no KVM

index	memoryVmUUID	memoryTotalMb	memoryUsedKB	memoryFreeKB
1	f4d32541 ...	128	131072	0
2	7cae0a2c ...	128	131072	0
3	a0216fad ...	64	65536	0
4	5c6bfe1 ...	128	131072	0
5	945c6a0e ...	128	131072	0
6	**ff74ed37 ...**	**128**	**131072**	**0**
7	0be03b74 ...	128	131072	0

Saída no Xen

index	memoryVmUUID	memoryTotalMb	memoryUsedKB	memoryFreeKB
1	0-0-0-0-0	2944	3014656	0
2	8f915a94 ...	1027	1052636	0
3	**738f9343...**	**128**	**131072**	**0**
4	e875600 ...	512	524288	0

Após alterar a quantidade de memória RAM, conforme o experimento de número 5, deverá ser conectado um determinado disco virtual na VM recém-criada. Essa atividade é

realizada através da tabela *storageTable*, que lista as unidade de armazenamento de todas as VM's. Inicialmente a VM recém-criada está conectada à imagem de disco indicada pelo modelo (*virtualMachineTemplateTable*). Essa imagem de disco deve ser substituída pela imagem de disco que deseja-se conectar à VM. O conteúdo de *storageTable* é apresentado na Tabela 5.8, onde as entradas correspondentes às VM recém-criadas aparecem em destaque. Para substituir seus discos virtuais, o valor da coluna *storageDiskImageIndex* deve ser substituído pelo valor de *diskImageIndex* de outra linha de *diskImagesTable*.

Tabela 5.8: Conteúdo de *StorageTable* no KVM e no Xen.

Saída no KVM				
storageId	storageVm UUID	storageDevice	storageDisk ImageIndex	storageRowStatus
1	f4d32541 ...	disk	6	active
2	f4d32541 ...	cdRom	0	active
3	7cae0a2c ...	disk	5	active
4	7cae0a2c ...	cdRom	0	active
5	a0216fad ...	disk	1	active
6	a0216fad ...	disk	2	active
7	a0216fad ...	cdRom	0	active
8	5c6bfe1 ...	disk	3	active
9	5c6bfe1 ...	cdRom	0	active
10	945c6a0e ...	disk	4	active
11	945c6a0e ...	cdRom	0	active
12	**ff74ed37 ...**	**disk**	**2**	**active**
13	ff74ed37 ...	cdRom	0	active
14	be03b74 ...	disk	7	active
15	be03b74 ...	cdRom	0	active
Saída no Xen				
storageId	storageVm UUID	storageDevice	storageDisk ImageIndex	storageRowStatus
1	8f915a94 ...	disk	2	active
2	8f915a94 ...	cdRom	0	active
3	**738f9343...**	**disk**	**4**	**active**
4	738f9343 ...	cdRom	0	active
5	e875600 ...	disk	3	active

O próximo experimento consiste em iniciar a VM recém-criada. Para alterar o estado de uma determinada VM, o valor do objeto *vmState* da tabela *virtualMachinesTable* deve ser alterado para o estado desejado. O objeto *vmState* admite os seguintes valores: 1 (*defined*), 2 (*running*), 3 (*blocked*), 4 (*paused*), 5 (*shutdown*), 6 (*shutoff*) e 7 (*crashed*). Para iniciar a VM recém criada, o valor de *vmState* na Tabela 5.6 deve ser configurado para 2 (*running*) na linha correspondente.

O experimento de número 7 da Tabela 5.3 consiste em inserir mais uma CPU vir-

tual na VM recém-criada. As CPU's das VM's são listadas em *cpuTable*. A VM recém criada contém inicialmente uma CPU virtual, conforme indicado pelo modelo. Para inserir mais uma CPU virtual na VM recém criada, o valor de *cpuRowStatus* da entrada correspondente em *cpuTable* deve ser alterado para 4 (*createAndGo*). A Tabela 5.9 apresenta o conteúdo de *cpuTable* com as entradas correspondentes às VM's recém-criadas em destaque, após a inserção de mais uma CPU em cada VM.

Tabela 5.9: Conteúdo de *CpuTable* no KVM e no Xen.

Saída no KVM					
index	cpuVmUUID	cpuArch	cpuMhz	cpuMilliseconds	cpuRowStatus
1	**ff74ed37 ...**	**x86_64**	**1197**	**2957518208**	**active**
2	**ff74ed37 ...**	**x86_64**	**1197**	**2957518208**	**active**
3	a0216fad ...	x86_64	1197	3910261632	active
4	0be03b74 ...	x86_64	1197	0	active
5	f4d32541 ...	x86_64	1197	0	active
6	7cae0a2c ...	x86_64	1197	0	active
7	5c6bfe1 ...	x86_64	1197	0	active
8	945c6a0e ...	x86_64	1197	0	active
Saída no Xen					
index	cpuVmUUID	cpuArch	cpuMhz	cpuMilliseconds	cpuRowStatus
1	0-0-0-0-0	x86_64	2494	3646051738	active
2	0-0-0-0-0	x86_64	2494	3646051738	active
3	0-0-0-0-0	x86_64	2494	3646051738	active
4	0-0-0-0-0	x86_64	2494	3646051738	active
5	8f915a94 ...	x86_64	2494	3733954217	active
6	**738f9343 ...**	**x86_64**	**2494**	**3047660344**	**active**
7	**738f9343 ...**	**x86_64**	**2494**	**3047660344**	**active**
8	0e875600 ...	x86_64	2494	48852610	active

Finalmente, o último experimento consiste em excluir a VM recém-criada. Para excluir uma VM, a linha correspondente de *virtualMachinesTable* (Tabela 5.6) deve ser apagada. Essa operação é executada através da alteração do valor do objeto *vmRowStatus* da linha de *virtualMachinesTable* correspondente para "6" (destroy). Com isso, a VM é excluída e seus recursos são liberados.

CAPÍTULO 6

CONCLUSÃO

Este trabalho apresentou uma estratégia para centralizar o gerenciamento de múltiplos monitores máquinas virtuais. A virtualização é uma tecnologia que permite executar mais de uma instância de sistema operacional em uma mesma máquina, ao mesmo tempo. Recentemente houve o surgimento e popularização da computação nas nuvens, ou *cloud computing*, um novo paradigma de organização e entrega de serviços através da Internet. A virtualização constitui a base da computação nas nuvens. A adoção da computação nas nuvens está impulsionando a integração de infraestruturas virtuais muitas vezes construídas sobre monitores de máquinas virtuais distintos.

Os principais monitores de máquinas virtuais existentes no mercado fornecem aplicativos próprios de gerência que, no entanto, são capazes de gerenciar somente o monitor de máquinas virtuais para o qual foram projetados. Além disso, não existe protocolo ou interface de gerência de VM's padronizada, impossibilitando a gerência de diferentes VMM's de modo centralizado e padronizado.

Este trabalho propõe a utilização do SNMP para a gerência de máquinas virtuais, compreendendo monitoramento e controle das VM's. Para isso, apresenta uma MIB chamada *Virtual-Machines-MIB*, que define uma interface padronizada para a gerência de máquinas virtuais. A MIB apresentada foi implementada com base no agente SNMP de domínio público NET-SNMP e na *libvirt*. O conjunto de funções oferecido pela *libvirt* não é capaz de atender a todas as funcionalidades da *Virtual-Machines-MIB*, sendo necessário utilizar outros meios, como as API's do sistema operacional hospedeiro e do VMM. A *Virtual-Machines-MIB* possui ainda uma tabela contendo modelos que descrevem uma VM, utilizados como base para a criação de novas VM's, fazendo com que o processo de criação de VM's seja completado de modo mais simples e direto.

Ao final, foram realizados experimentos que consistiram em diversas operações de

monitoramento e controle sobre os monitores de máquinas virtuais KVM e Xen. Essas operações foram realizadas através de ferramentas de gerência SNMP comuns, a partir de uma única estação realizando a gerência de dois VMM's distintos, demonstrando assim o gerenciamento centralizado de múltiplos VMM's através da *Virtual-Machines-MIB*.

Trabalhos futuros incluem a inclusão de novos objetos de gerência na MIB, por exemplo, para a criação e gerência de imagens de disco, para a mudança da ordem de inicialização das VM's e para gerência de redes virtuais. Também deve ser implementado o suporte a outros VMM's, assim como ser realizada uma avaliação do desempenho da solução.

REFERÊNCIAS

[1] BIRD: Binary Interpretation using Runtime Disassembly. *http://www.ecsl.cs.sunysb.edu/bird/index.html*. Acessado em 12/10/2011.

[2] Bochs x86 PC Emulator Users Manual. *http://bochs.sourceforge.net*. Acessado em 12/10/2011.

[3] Cacti: The Complete RRDTool-based Graphing Solution. *http://www.cacti.net/*. Acessado em 15/03/2012.

[4] Cisco Fabric Manager. *http://www.cisco.com/en/US/products/ps10495/index.html*. Acessado em 03/11/2011.

[5] Cricket Home. *http://cricket.sourceforge.net/*. Acessado em 15/03/2012.

[6] Homepage of Zabbix: An Enterprise-Class Open Source Distributed Monitoring Solution. *http://www.zabbix.com/*. Acessado em 15/03/2012.

[7] HP Network Node Manager. *http://www8.hp.com/us/en/software/software-product.html?compURI=tcm:245-936950&pageTitle=network-node-manager-i-software*. Acessado em 03/11/2011.

[8] IBM Tivoli software. *http://www-01.ibm.com/software/tivoli/*. Acessado em 03/11/2011.

[9] Libvirt, the Virtualization API. *http://libvirt.org/*. Acessado em 07/08/2012.

[10] libvirt: Wiki: Libvirt-snmp. *http://wiki.libvirt.org/page/Libvirt-snmp*. Acessado em 21/08/2012.

[11] Microsoft Hyper-V Server. *http://www.microsoft.com/hyper-v-server/*. Acessado em 07/08/2012.

[12] Open Cloud Computing Interface. *http://occi-wg.org/*. Acessado em 21/08/2012.

[13] Qemu cpu emulator. *http://qemu.org*. Acessado em 12/10/2011.

[14] System Virtualization Profile, DMTF profile DSP1042 v. 1.0.0. *http://dmtf.org/sites/default/files/standards/documents/DSP1042_1.0.0_0.pdf*. Acessado em 07/08/2012.

[15] The NET-SNMP Home Page. *http://www.net-snmp.org/*. Acessado em 07/08/2012.

[16] Tobi Oetiker's MRTG - The Multi Router Traffic Grapher. *http://oss.oetiker.ch/mrtg/*. Acessado em 15/03/2012.

[17] TUBS-IBR-XEN-MIB DEFINITIONS. *http://www.ibr.cs.tu-bs.de/svn/libsmi/trunk/mibs/tubs/TUBS-IBR-XEN-MIB*. Acessado em 07/08/2012.

[18] Virtual Machine Manager. *http://virt-manager.org/*. Acessado em 17/01/2012.

[19] Virtual System Profile, DMTF profile DSP1057 v. 1.0.0. *http://dmtf.org/sites/default/files/standards/documents/DSP1057_1.0.0_0.pdf*. Acessado em 07/08/2012.

[20] VMWare. *http://www.vmware.com*. Acessado em 07/08/2012.

[21] Web Interface to libsmi Tools. *http://www.ibr.cs.tu-bs.de/projects/libsmi/tools/*. Acessado em 21/08/2012.

[22] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, e I. Pratt. Xen and the Art of Virtualization. *Proc. of the 19th ACM Symp. on Operating Systems Principles*, páginas 164–177, outubro de 2003.

[23] A. Bianco, R. Birke, F.G. Debele, e L. Giraudo. SNMP Management in a Distributed Software Router Architecture. *Proc of the IEEE Intl. Conference on Communications (ICC'11)*, páginas 1–5, junho de 2011.

[24] M. Bolte, M. Sievers, G. Birkenheuer, O. Niehörster, e A. Brinkmann. Non-Intrusive Virtualization Management Using Libvirt. *Proc. of the Design, Automation and Test in Europe Conf. and Exhibition (DATE'10)*, páginas 574–579, março de 2010.

[25] J. Case, M. Fedor, M. Schoffstall, e J. Davin. Simple Network Management Protocol. *RFC 1157*, Maio de 1990.

[26] J. Case, K. McCloghrie, M. Rose, e S. Waldbusser. Coexistence between version 1 and version 2 of the Internet-standard Network Management Framework. *RFC 1452*, Abril de 1993.

[27] J. Case, K. McCloghrie, M. Rose, e S. Waldbusser. Introduction to version 2 of the Internet-standard Network Management Framework. *RFC 1441*, Abril de 1993.

[28] F. Daitx, F. Daitx, R. P. Esteves, e L. Z. Granville. On the Use of SNMP as a Management Interface for Virtual Networks. *Proc. of the IFIP/IEEE Intl. Symp. on Integrated Network Management (IM'11)*, páginas 177–184, maio de 2011.

[29] V. A. Danciu. Host Virtualization: a Taxonomy of Management Challenges. *Proc. of the 2nd Workshop on Services, Platforms, Innovations and Research for new Infrastructures in Telecommunications (SPIRIT'09)*, outubro de 2009.

[30] P. Goncalves, J.L. Oliveira, e R.L. Aguiar. An evaluation of network management protocols. *Proc. of the IFIP/IEEE International Symposium on Integrated Network Management (IM'09)*, páginas 537–544, junho de 2009.

[31] S. Graupner, R. König, V. Machiraju, J. Pruyne, A. Sahai, e A. Moorsel. Impact of Virtualization on Management Systems. Relatório Técnico HPL-2003-125, Hewlett-Packard, 2003.

[32] D Gupta, L Cherkasova, R Gardner, e A Vahdat. Enforcing performance isolation across virtual machines in Xen. *Proc. of the ACM/IFIP/USENIX 2006 Intl. Conf. on Middleware*, páginas 342–362, 2006.

[33] I. Habib. Virtualization with kvm. *Linux Journal*, 2008(166):8, 2008.

[34] H. Han, S. Kim, H. Jung, H.Y. Yeom, C. Yoon, J. Park, e Y. Lee. A RESTful Approach to the Management of Cloud Infrastructure. *Proc. of the IEEE Intl. Conf. on Cloud Computing (CLOUD'09)*, páginas 139–142, setembro de 2009.

[35] S. Harnedy. *Total SNMP: Exploring the Simple Network Management Protocol*. Prentice Hall PTR, 2nd edition, 1997.

[36] D. Harrington, R. Presuhn, e B. Wijnen. An Architecture for Describing SNMP Management Frameworks. *RFC 2271*, janeiro de 1998.

[37] A. Klaiber. The Technology Behind Crusoe Processors. Relatório técnico, Transmeta Corporation, 2000.

[38] P. Leach, M. Mealling, e R. Salz. A Universally Unique IDentifier (UUID) URN Namespace. *RFC 4122*, julho de 2005.

[39] B. Ligneris. Virtualization of Linux based computers: the Linux-VServer project. *Proc. of the 19th Intl. Symp. on High Performance Computing Systems and Applications (HPCS'05)*, páginas 340–346, maio de 2005.

[40] K. McCloghrie, D. Perkins, e J. Schoenwaelder. Structure of Management Information Version 2 (SMIv2). *RFC 2578*, abril de 1999.

[41] S. Nanda e T. Chiueh. A Survey on Virtualization Technologies. Relatório Técnico TR-179, Stony Brook Univ., fevereiro de 2005.

[42] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, e D. Zagorodnov. The Eucalyptus Open-Source Cloud-Computing System. *Proc. of the 9th IEEE/ACM International Symp. on Cluster Computing and the Grid (CCGRID'09)*, páginas 124–131, maio de 2009.

[43] Y. Peng e Y. Chen. SNMP-based Monitoring of Heterogeneous Virtual Infrastructure in Clouds. *Proc. of the 13th Asia-Pacific Network Operations and Management Symp. (APNOMS'11)*, páginas 1–6, setembro de 2011.

[44] G. J. Popek e R. P. Goldberg. Formal requirements for virtualizable third generation architectures. *Communications of the ACM*, 17:412–421, julho de 1974.

[45] M. Rose e K. McCloghrie. Structure and Identification of Management Information for TCP/IP-based internets. *RFC 1155*, maio de 1990.

[46] M. Rose. e K. McCloghrie. Management Information Base for Network Management of TCP/IP-based internets: MIB-II. *RFC 1213*, Março de 1991.

[47] E. Salvador, J. Almeida, J.M. Nogueira, P. Barbosa, e L. Granville. A characterization study of SNMP usage patterns. *Proc. of the IFIP/IEEE Intl. Symp. on Integrated Network Management (IM'11)*, páginas 690–693, maio de 2011.

[48] M. Savic, S. Gajin, e M. Bozic. SNMP-based Grid Infrastructure Monitoring System. *Proc. of the 34th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO'11)*, páginas 231–235, maio de 2011.

[49] B. Sotomayor, R. S. Montero, I. M. Llorente, e I. Foster. Capacity Leasing in Cloud Systems Using the Opennebula Engine. *Proc. of the Workshop on Cloud Computing and its Applications (CCA'08)*, 2008.

[50] B. Sotomayor, R. S. Montero, I. M. Llorente, e I. Foster. Virtual Infrastructure Management on Private and Hybrid Clouds. *IEEE Internet Computing*, 13(5):14–22, 2009.

[51] W. Stallings. *Snmp, Snmpv2 and Rmon*. Addison-Wesley, Reading, MA, 2nd edition, 1996.

[52] W. Stallings. Security Comes to SNMP: The New SNMPv3 Proposed Internet Standards. *The Internet Protocol Journal*, 1:2–12, dezembro de 1998.

[53] A. Whitaker, M. Shaw, e S. Gribble. Scale and Performance in the Denali isolation kernel. *Proc. of the ACM Operation Systems Review (OSDI'02)*, páginas 195–210, 2002.

[54] S. Wu, L. Deng, H. Jin, X. Shi, Y. Zhao, W. Gao, J. Zhang, e J. Peng. Virtual Machine Management Based on Agent Service. *Proc. of the 11th Intl. Conf. on Parallel and Distributed Computing, Applications and Technologies (PDCAT'10)*, páginas 199–204, dezembro de 2010.

[55] Q. Zhang, L. Cheng, e R. Boutaba. Cloud computing: State-of-the-art and Research Challenges. *Journal of Internet Services and Applications*, 1:7–18, 2010.

APÊNDICE

Descrição ASN.1 Virtual-Machines-MIB

```
VIRTUAL-MACHINES-MIB DEFINITIONS ::= BEGIN

IMPORTS
MODULE-IDENTITY, OBJECT-TYPE, enterprises, Unsigned32,
Counter32, IpAddress
FROM SNMPv2-SMI
OBJECT-GROUP, MODULE-COMPLIANCE
FROM SNMPv2-CONF
TEXTUAL-CONVENTION, RowStatus, DisplayString, PhysAddress
FROM SNMPv2-TC
;

virtualMachinesMIB MODULE-IDENTITY
LAST-UPDATED "201203130000Z"
ORGANIZATION "Federal University of Parana - Dept. Informatics"
CONTACT-INFO
"Ricardo Hillbrecht
Luis Carlos Erpen de Bona
Federal University of Parana
Dept. Informatics
P.O. Box 19081
Curitiba, PR 81531-980
Brazil
Phone +55-41-3361-3031
Email: {ricardoh, bona}@inf.ufpr.br"
DESCRIPTION
"This MIB module define a set of objects
to manage virtual machines."
REVISION    "201203130000Z"
    DESCRIPTION
        "The initial revision of this module."
::= { yyy xxx } -- to be assigned by IANA


virtualMachines     OBJECT IDENTIFIER ::= { virtualMachinesMIB 1 }
virtualMachinesMIBConformance     OBJECT IDENTIFIER ::= { virtualMachinesMIB 2 }

-- new types
UUID ::= TEXTUAL-CONVENTION
    DISPLAY-HINT "4x-2x-2x-2x-6x"
    STATUS      current
    DESCRIPTION
"The Universally Unique IDentifier (UUID) as defined by RFC 4122."
    SYNTAX     OCTET STRING (SIZE (16))

WWN ::= TEXTUAL-CONVENTION
    DISPLAY-HINT "1x:"
    STATUS       current
```

```
    DESCRIPTION
            "A World Wide Name (WWN) is a unique identifier
in a Serial Attached SCSI storage network or
Fibre Channel. Each WWN is an 8-byte number
derived from an IEEE OUI (for the first 3 bytes)
and vendor-supplied information."
    SYNTAX       OCTET STRING (SIZE (8))

--Disk Images Table

diskImagesTable OBJECT-TYPE
    SYNTAX     SEQUENCE OF DiskImagesEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
        "Contains disk image files that can be used
as storage devices by the virtual machines."
    ::= { virtualMachines 1 }

diskImagesEntry OBJECT-TYPE
    SYNTAX     DiskImagesEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
        "A (conceptual) entry for one disk image file."
    INDEX { diskImageIndex }
    ::= { diskImagesTable 1 }

DiskImagesEntry ::= SEQUENCE {
diskImageIndex Unsigned32,
diskImageId Unsigned32,
diskImageName DisplayString,
diskImageFileName DisplayString,
diskImageTotalMb Unsigned32,
diskImageUsedMb Unsigned32,
diskImageAvailMb Unsigned32,
diskImageType DisplayString
}

diskImageIndex OBJECT-TYPE
    SYNTAX     Unsigned32
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
        "A unique value for each disk image file."
    ::= { diskImagesEntry 1 }

diskImageId OBJECT-TYPE
    SYNTAX     Unsigned32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
        "Same value as diskImageIndex, note that indexes in SMIv2 are not accessible."
    ::= { diskImagesEntry 2 }

diskImageName OBJECT-TYPE
    SYNTAX     DisplayString
    MAX-ACCESS read-only
```

```
    STATUS     current
    DESCRIPTION
        "A description of the disk image file contained by this entry."
    ::= { diskImagesEntry 3 }

diskImageFileName OBJECT-TYPE
    SYNTAX     DisplayString
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
        "Contains the name of the disk image file."
    ::= { diskImagesEntry 4 }

diskImageTotalMb OBJECT-TYPE
    SYNTAX     Unsigned32
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
        "Contains the size of the disk image file, in MB."
    ::= { diskImagesEntry 5 }

diskImageUsedMb OBJECT-TYPE
    SYNTAX      Unsigned32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
"Used portion of storage device, in MB."
    ::= { diskImagesEntry 6 }

diskImageAvailMb OBJECT-TYPE
    SYNTAX      Unsigned32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
"Amount of memory available in the storage device, in MB."
    ::= { diskImagesEntry 7 }

diskImageType OBJECT-TYPE
    SYNTAX     DisplayString
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
        "Contains the file type of the disk image, e.g: RAW, QCOW, VDI, VMDK, VHD, HDD, etc."
    ::= { diskImagesEntry 8 }

--Virtual Machines Table

virtualMachinesTable OBJECT-TYPE
    SYNTAX     SEQUENCE OF VirtualMachinesEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
        "Contains the virtual machines."
    ::= { virtualMachines 2 }

virtualMachinesEntry OBJECT-TYPE
    SYNTAX     VirtualMachinesEntry
    MAX-ACCESS not-accessible
```

```
    STATUS     current
    DESCRIPTION
        "A (conceptual) entry for one VM."
    INDEX { vmIndex }
    ::= { virtualMachinesTable 1 }

VirtualMachinesEntry ::= SEQUENCE {
vmIndex Unsigned32,
vmId Unsigned32,
vmUUID UUID,
vmName DisplayString,
vmOsIndex Unsigned32,
vmState INTEGER,
vmConfFile DisplayString,
vmKernelIndex Unsigned32,
vmRowStatus RowStatus
}

vmIndex OBJECT-TYPE
    SYNTAX      Unsigned32
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
"A unique value for each virtual machine."
    ::= { virtualMachinesEntry 1 }

vmId OBJECT-TYPE
    SYNTAX      Unsigned32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
"Same value as vmIndex, note that
indexes in SMIv2 are not accessible."
    ::= { virtualMachinesEntry 2 }

vmUUID OBJECT-TYPE
    SYNTAX      UUID
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
"The UUID of the virtual machine."
    ::= { virtualMachinesEntry 3 }

vmName OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
"Contains the name of the virtual machine."
    ::= { virtualMachinesEntry 4 }

vmOsIndex OBJECT-TYPE
    SYNTAX      Unsigned32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
"Contains the identifier of guest operating
system from SupportedOsTable."
```

```
    ::= { virtualMachinesEntry 5 }

vmState OBJECT-TYPE
    SYNTAX      INTEGER {
defined(1),
    running(2),
    blocked(3),
    paused(4),
    shutdown(5),
    shutoff(6),
    crashed(7)
    }
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
"Current state of the guest. This column is
writable and permits to change the guest state.
The states that can be assumed by the guest are:
defined(1)
    running(2)
    blocked(3)
    paused(4)
    shutdown(5)
    shutoff(6)
    crashed(7)"
    ::= { virtualMachinesEntry 6}

vmConfFile OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
"Contains the name of the configuration file of a virtual machine."
    ::= { virtualMachinesEntry 7 }

vmKernelIndex OBJECT-TYPE
    SYNTAX      Unsigned32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
"Contains the identifier of the kernel of
guest operating system, from KernelTable."
    ::= { virtualMachinesEntry 8 }

vmRowStatus OBJECT-TYPE
SYNTAX      RowStatus
MAX-ACCESS  read-create
STATUS      current
DESCRIPTION
"The status of the vm.
A new virtual machine can be created by
setting this column to 'createAndGo'.
A virtual machine can be destroyed by
setting this column value to 'destroy'."
DEFVAL  { active }
::= { virtualMachinesEntry 9 }

-- Supported OS Table
```

```
supportedOsTable OBJECT-TYPE
    SYNTAX     SEQUENCE OF SupportedOsEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
        "Contain informations about the guest operating
systems supported by the hypervisor."
    ::= { virtualMachines 3 }

supportedOsEntry OBJECT-TYPE
    SYNTAX     SupportedOsEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
        "A (conceptual) entry for one supported OS."
    INDEX { osIndex }
    ::= { supportedOsTable 1 }

SupportedOsEntry ::= SEQUENCE {
osIndex Unsigned32,
osId Unsigned32,
osVendor DisplayString,
osName DisplayString,
osVersion DisplayString,
osBuild DisplayString
}

osIndex OBJECT-TYPE
    SYNTAX      Unsigned32
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
"A unique value for each operating system."
    ::= { supportedOsEntry 1 }

osId OBJECT-TYPE
    SYNTAX      Unsigned32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
"Same value as osIndex, note that
indexes in SMIv2 are not accessible."
    ::= { supportedOsEntry 2 }

osVendor OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
"Vendor of the operating system."
    ::= { supportedOsEntry 3 }

osName OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
```

```
"Name of the operating system."
    ::= { supportedOsEntry 4 }

osVersion OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
"Version of the operating system."
    ::= { supportedOsEntry 5 }

osBuild OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
"Build data about the operating system."
    ::= { supportedOsEntry 6 }

--VirtualMachineMonitor group

virtualMachineMonitor     OBJECT IDENTIFIER ::= { virtualMachines 4 }

vmmName OBJECT-TYPE
    SYNTAX     DisplayString
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
        "Virtual machines monitor name."
    ::= { virtualMachineMonitor 1 }

vmmVersion OBJECT-TYPE
    SYNTAX     DisplayString
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
        "Version of virtual machines monitor."
    ::= { virtualMachineMonitor 2 }


-- Kernel Table

kernelTable OBJECT-TYPE
    SYNTAX     SEQUENCE OF KernelEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
        "Contains operating systems kernels
available to the virtual machines."
    ::= { virtualMachines 5 }

kernelEntry OBJECT-TYPE
    SYNTAX     KernelEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
        "A (conceptual) entry for one supported SO kernel."
    INDEX { kernelIndex }
```

```
    ::= { kernelTable 1 }

KernelEntry ::= SEQUENCE {
kernelIndex Unsigned32,
kernelId Unsigned32,
kernelName DisplayString,
kernelPath DisplayString,
kernelParams DisplayString
}

kernelIndex OBJECT-TYPE
    SYNTAX      Unsigned32
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
"A unique value for each operating system kernel."
    ::= { kernelEntry 1 }

kernelId OBJECT-TYPE
    SYNTAX      Unsigned32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
"Same value as kernelIndex, note that
indexes in SMIv2 are not accessible."
    ::= { kernelEntry 2 }

kernelName OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
"Contains the name of the operating system kernel."
    ::= { kernelEntry 3 }

kernelPath OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
"Contains the path to the kernel file."
    ::= { kernelEntry 4 }

kernelParams OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
"Parameters passed to the kernel at startup."
    ::= { kernelEntry 6 }

-- Storage Table

storageTable OBJECT-TYPE
    SYNTAX     SEQUENCE OF StorageEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
```

```
        "Contains the storage devices being used by
the virtual machines (relates the image files
of DiskImagesTable with one or more virtual
machines of VirtualMachinesTable)."
    ::= { virtualMachines 6 }

storageEntry OBJECT-TYPE
    SYNTAX     StorageEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
        "A (conceptual) entry for every alocated storage area."
INDEX { storageIndex }
    ::= { storageTable 1 }

StorageEntry ::= SEQUENCE {
storageIndex Unsigned32,
storageId Unsigned32,
storageVmUUID UUID,
storageDevice INTEGER,
storageDiskImageIndex Unsigned32,
storageRowStatus RowStatus
}

storageIndex OBJECT-TYPE
    SYNTAX      Unsigned32
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
"A unique value for each storage device."
    ::= { storageEntry 1 }

storageId OBJECT-TYPE
    SYNTAX      Unsigned32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
"Same value as storageIndex, note that
indexes in SMIv2 are not accessible."
    ::= { storageEntry 2 }

storageVmUUID OBJECT-TYPE
    SYNTAX      UUID
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
"The UUID of the virtual guest
(from virtualMachinesTable)."
    ::= { storageEntry 3 }

storageDevice OBJECT-TYPE
    SYNTAX     INTEGER {
disk(1),
cdRom(2),
other(3)
    }
    MAX-ACCESS read-only
    STATUS     current
```

```
    DESCRIPTION
"The type of storage device. Possible storage devices types are:
disk(1)
cdRom(2)
other(3)"
    ::= { storageEntry 4 }

storageDiskImageIndex OBJECT-TYPE
    SYNTAX     Unsigned32
    MAX-ACCESS read-write
    STATUS     current
    DESCRIPTION
        "Contains the index of an entry in DiskImagesTable
that represents the backend file of this storage device."
    ::= { storageEntry 5 }

storageRowStatus OBJECT-TYPE
        SYNTAX      RowStatus
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
                "The status of the storage entry."
        DEFVAL  { active }
        ::= { storageEntry 6 }

-- Memory Table

memoryTable OBJECT-TYPE
    SYNTAX     SEQUENCE OF MemoryEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
        "Contains data related to virtual
machine RAM utilization, like total,
used and available amount of RAM."
    ::= { virtualMachines 7 }

memoryEntry OBJECT-TYPE
    SYNTAX     MemoryEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
        "A (conceptual) entry for every VM
containing data about RAM utilization."
    INDEX { memoryIndex }
    ::= { memoryTable 1 }

MemoryEntry ::= SEQUENCE {
memoryIndex Unsigned32,
memoryVmUUID UUID,
memoryTotalMb Unsigned32,
memoryUsedKB Unsigned32,
memoryFreeKB Unsigned32
}

memoryIndex OBJECT-TYPE
    SYNTAX      Unsigned32
    MAX-ACCESS  not-accessible
```

```
    STATUS      current
    DESCRIPTION
"A unique value for each virtual machine RAM."
    ::= { memoryEntry 1 }

memoryVmUUID OBJECT-TYPE
    SYNTAX      UUID
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
"The UUID of the virtual guest
(from virtualMachinesTable)."
    ::= { memoryEntry 2 }

memoryTotalMb OBJECT-TYPE
    SYNTAX      Unsigned32
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
"Total amount of RAM available to the VM."
    ::= { memoryEntry 3 }

memoryUsedKB OBJECT-TYPE
    SYNTAX      Unsigned32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The amount of memory in Kbytes currently occupied
by the virtual machine."
    ::= { memoryEntry 4 }

memoryFreeKB OBJECT-TYPE
    SYNTAX      Unsigned32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The amount of memory in Kbytes currently available
to the virtual machine."
    ::= { memoryEntry 5 }

-- CPU Table

cpuTable OBJECT-TYPE
    SYNTAX     SEQUENCE OF CpuEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
        "Contains data related with the virtual
machine CPUs, like number of virtual CPUs,
instruction set architecture, clock frequency
and milliseconds consumed by the virtual CPU."
    ::= { virtualMachines 8 }

cpuEntry OBJECT-TYPE
    SYNTAX     CpuEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
```

```
        "An entry describing a virtual CPU of a VM."
    INDEX { cpuIndex }
    ::= { cpuTable 1 }

CpuEntry ::= SEQUENCE {
cpuIndex Unsigned32,
cpuVmUUID UUID,
cpuArch DisplayString,
cpuMhz Unsigned32,
cpuMilliseconds Counter32,
cpuRowStatus RowStatus
}

cpuIndex OBJECT-TYPE
    SYNTAX      Unsigned32
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
"A unique value for each CPU of a virtual machine."
    ::= { cpuEntry 1 }

cpuVmUUID OBJECT-TYPE
    SYNTAX      UUID
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
"The UUID of the virtual guest
(from virtualMachinesTable)."
    ::= { cpuEntry 2 }

cpuArch OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
"The instruction set architecture of virtual CPU."
    ::= { cpuEntry 3 }

cpuMhz OBJECT-TYPE
    SYNTAX      Unsigned32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
       "The frequency in MHz of the virtual CPU."
    ::= { cpuEntry 4 }

cpuMilliseconds OBJECT-TYPE
    SYNTAX      Counter32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of milliseconds consumed by
the virtual CPU since the virtual machine
has been started."
    ::= { cpuEntry 5 }

cpuRowStatus OBJECT-TYPE
SYNTAX      RowStatus
```

```
MAX-ACCESS  read-create
STATUS      current
DESCRIPTION
"The status of the cpu entry."
DEFVAL  { active }
::= { cpuEntry 6 }

-- HBA Table

hbaTable OBJECT-TYPE
    SYNTAX     SEQUENCE OF HbaEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
        "Table of host bus adapters (hba) for all vms in virtualMachinesTable."
    ::= { virtualMachines 9 }

hbaEntry OBJECT-TYPE
    SYNTAX     HbaEntry
    MAX-ACCESS not-accessible
    STATUS     current
    DESCRIPTION
        "Uniquely identifies a given virtual machine host bus adapter."
    INDEX { hbaIndex }
    ::= { hbaTable 1 }

HbaEntry ::= SEQUENCE {
hbaIndex Unsigned32,
hbaVmUUID UUID,
hbaName DisplayString,
hbaVirtDev DisplayString,
hbaWwn WWN
}

hbaIndex OBJECT-TYPE
    SYNTAX      Unsigned32
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
"Uniquely identifies a given Host Bus adapter in this VM."
    ::= { hbaEntry 1 }

hbaVmUUID OBJECT-TYPE
    SYNTAX      UUID
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
"The UUID of the virtual guest
(from virtualMachinesTable)."
    ::= { hbaEntry 2 }

hbaName OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
"The system device name for this host bus adapter."
    ::= { hbaEntry 3 }
```

```
hbaVirtDev OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
"The oem host bus adapter hardware being emulated to the Guest OS."
    ::= { hbaEntry 4 }

hbaWwn OBJECT-TYPE
    SYNTAX      WWN
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
"The 8-byte address assigned to a
Host Adapter Bus (HBA) in a Fibre
Channel network."
    ::= { hbaEntry 5 }

-- Network table:

networkTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF NetworkEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "A list of all networks per virtual machine."
    ::= { virtualMachines 10 }

networkEntry OBJECT-TYPE
    SYNTAX      NetworkEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "An entry describing a network of a virtual machine."
    INDEX   { networkIndex }
    ::= { networkTable 1 }

NetworkEntry ::= SEQUENCE {
networkIndex         Unsigned32,
networkVmUUID UUID,
networkDescr DisplayString,
networkInOctets Counter32,
networkInPackets Counter32,
networkInErrors Counter32,
networkInDiscards Counter32,
networkOutOctets Counter32,
networkOutPackets Counter32,
networkOutErrors Counter32,
networkOutDiscards Counter32,
interfacePhysAddress PhysAddress,
networkIpAddress IpAddress,
networkIpMask IpAddress,
networkBroadcast IpAddress,
networkGateway IpAddress,
networkLocalNet IpAddress
}
```

```
networkIndex OBJECT-TYPE
    SYNTAX  Unsigned32
    MAX-ACCESS  not-accessible
    STATUS  current
    DESCRIPTION
"A unique value for each network interface."
    ::= { networkEntry 1 }

networkVmUUID OBJECT-TYPE
    SYNTAX      UUID
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
"The UUID of the virtual guest
(from virtualMachinesTable)."
    ::= { networkEntry 2 }

networkDescr OBJECT-TYPE
    SYNTAX  DisplayString
    MAX-ACCESS  read-write
    STATUS  current
    DESCRIPTION
"A textual string containing information about the interface."
    ::= { networkEntry 3 }

networkInOctets OBJECT-TYPE
    SYNTAX  Counter32
    MAX-ACCESS  read-only
    STATUS  current
    DESCRIPTION
"The total number of octets received on
the interface since the virtual machine
has been set up."
    ::= { networkEntry 4 }

networkInPackets OBJECT-TYPE
    SYNTAX      Counter32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of packets received on the network
interface since the virtual machine has been set up."
    ::= { networkEntry 5 }

networkInErrors OBJECT-TYPE
    SYNTAX      Counter32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of erroneous packets received on the network
interface since the virtual machine has been set up."
    ::= { networkEntry 6 }

networkInDiscards OBJECT-TYPE
    SYNTAX      Counter32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
```

```
        "The number of dropped packets received on the network
interface since the virtual machine has been set up."
    ::= { networkEntry 7 }

networkOutOctets OBJECT-TYPE
    SYNTAX      Counter32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of octets sent on the network
interface since the virtual machine has been set up."
    ::= { networkEntry 8 }

networkOutPackets OBJECT-TYPE
    SYNTAX      Counter32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of packets sent on the network
interface since the virtual machine has been set up."
    ::= { networkEntry 9 }

networkOutErrors OBJECT-TYPE
    SYNTAX      Counter32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of packets that could not be sent
on the network interface because of any errors
since the virtual machine has been set up."
    ::= { networkEntry 10 }

networkOutDiscards OBJECT-TYPE
    SYNTAX      Counter32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of packets that have not been sent
 on the network interface even though no errors
 had been detected since the virtual machine
 has been set up."
    ::= { networkEntry 11 }

interfacePhysAddress OBJECT-TYPE
    SYNTAX  PhysAddress
    MAX-ACCESS  read-only
    STATUS  current
    DESCRIPTION
            "The interface's address at the protocol layer
            immediately 'below' the network layer in the
            protocol stack.  For interfaces which do not have
            such an address, this object should contain an
DisplayString of zero length."
    ::= {  networkEntry 12 }

networkIpAddress OBJECT-TYPE
    SYNTAX  IpAddress
    MAX-ACCESS  read-only
```

```
    STATUS  current
    DESCRIPTION
            "The IP address associated to the interface."
    ::= { networkEntry 13 }

networkIpMask OBJECT-TYPE
    SYNTAX  IpAddress
    MAX-ACCESS  read-only
    STATUS  current
    DESCRIPTION
            "The subnet mask associated with the IP address of
            this interface."
    ::= { networkEntry 14 }

networkBroadcast OBJECT-TYPE
    SYNTAX  IpAddress
    MAX-ACCESS  read-only
    STATUS  current
    DESCRIPTION
            "The broadcast IP address of the network."
    ::= { networkEntry 15 }

networkGateway OBJECT-TYPE
    SYNTAX  IpAddress
    MAX-ACCESS  read-only
    STATUS  current
    DESCRIPTION
            "The IP address of the network gateway."
    ::= { networkEntry 16 }

networkLocalNet OBJECT-TYPE
    SYNTAX  IpAddress
    MAX-ACCESS  read-only
    STATUS  current
    DESCRIPTION
            "The IP address of the local network."
    ::= { networkEntry 17 }

--virtualMachinesMibTemplates group

virtualMachinesMibTemplates OBJECT IDENTIFIER ::= { virtualMachines 11 }

virtualMachineTemplateTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF VirtualMachineTemplateEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "A table containing the template used to create a new virtual machine."
    ::= { virtualMachinesMibTemplates 1 }

virtualMachineTemplateEntry OBJECT-TYPE
    SYNTAX      VirtualMachineTemplateEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "An entry describing a template used to create a new virtual machines."
    INDEX   { virtualMachineTemplateIndex }
    ::= { virtualMachineTemplateTable 1 }
```

```
VirtualMachineTemplateEntry ::= SEQUENCE {
virtualMachineTemplateIndex Unsigned32,
virtualMachineTemplateId Unsigned32,
domainTemplateType DisplayString,
domainTemplateName DisplayString,
domainTemplateMemory Unsigned32,
domainTemplateVcpus Unsigned32,
bootOsType DisplayString,
bootOsDev DisplayString,
domainClockOffset DisplayString,
deviceEmulator DisplayString,
diskSource DisplayString,
domainMacAddress DisplayString,
virtualNetwork DisplayString,
graphicsType DisplayString,
soundModel DisplayString,
virtualMachineTemplateRowStatus RowStatus
}

virtualMachineTemplateIndex OBJECT-TYPE
    SYNTAX  Unsigned32
    MAX-ACCESS  not-accessible
    STATUS  current
    DESCRIPTION
"A unique value for each entry."
    ::= { virtualMachineTemplateEntry 1 }

virtualMachineTemplateId OBJECT-TYPE
    SYNTAX  Unsigned32
    MAX-ACCESS read-only
    STATUS  current
    DESCRIPTION
"Same value as virtualMachineTemplateIndex ,
note that indexes in SMIv2 are not accessible."
    ::= { virtualMachineTemplateEntry 2 }

domainTemplateType OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
"Value of type attribute of XML domain tag,
as defined in http://www.libvirt.org/formatdomain.html.
Specifies the hypervisor used for running the domain.
The allowed values include xen, kvm, qemu, lxc and kqemu."
    ::= { virtualMachineTemplateEntry 3 }

domainTemplateName OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
"Name of vm template."
    ::= { virtualMachineTemplateEntry 4 }

domainTemplateMemory OBJECT-TYPE
    SYNTAX      Unsigned32
```

```
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
"The amount of memory for the guest at boot time,
in KiB (kibibytes, 2^10 or blocks of 1024 bytes)."
    ::= { virtualMachineTemplateEntry 5}

domainTemplateVcpus OBJECT-TYPE
    SYNTAX      Unsigned32
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
"Number of virtual CPUs allocated for the guest OS,
which must be between 1 and the maximum supported by the hypervisor."
    ::= { virtualMachineTemplateEntry 6}

bootOsType OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
"Specifies the type of operating system to be booted in the virtual machine.
The value of this object is used in tag type, child of tag os in a domain XML definition.
Alowed values are hvm for full virtualized guest and linux for paravirtualized guest.
Example:
<os>
<type/>hvm</type>
</os>"
    ::= { virtualMachineTemplateEntry 7}

bootOsDev OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
"dev attribute takes one of the values fd, hd, cdrom or
network and is used to specify a boot device.
The value of this object is used in dev attribute of tag boot,
child of tag os in a domain XML definition.
Example:
<os>
<boot dev='hd'/>
</os>"
::= { virtualMachineTemplateEntry 8}

domainClockOffset OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
"The offset attribute of tag clock takes four possible values
(utc, localtime, timezone or variable) allowing fine grained
control over how the guest clock is synchronized to the host."
::= { virtualMachineTemplateEntry 9}

deviceEmulator OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  read-write
```

```
    STATUS      current
    DESCRIPTION
"Specify the fully qualified path to the device model emulator binary."
::= { virtualMachineTemplateEntry 10}

diskSource OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
"Specifies the fully-qualified path to the file holding the disk."
::= { virtualMachineTemplateEntry 11}

domainMacAddress OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
"MAC address to be used in ethernet interface of a domain template."
::= { virtualMachineTemplateEntry 12}

virtualNetwork OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
"Virtual network name which the new vm will
be connected when the vm template be instantiated."
::= { virtualMachineTemplateEntry 13}

graphicsType OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
"A graphics device allows for graphical interaction with the guest OS.
Allowed values are sdl, vnc, rdp or desktop"
::= { virtualMachineTemplateEntry 14}

soundModel OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
"Specifies what real sound device is emulated.
Typical values are es1370, sb16, ac97, and ich6."
::= { virtualMachineTemplateEntry 15}

virtualMachineTemplateRowStatus OBJECT-TYPE
SYNTAX      RowStatus
MAX-ACCESS  read-create
STATUS      current
DESCRIPTION
"The status of the template entry."
DEFVAL  { active }
::= { virtualMachineTemplateEntry 16 }
```

```
storageTemplate OBJECT-TYPE
    SYNTAX     Unsigned32
    MAX-ACCESS read-write
    STATUS     current
    DESCRIPTION
        "Disk image file used as template to create
a new storage entry in storageTable. Reffers
to diskImageIndex field of diskImagesTable."
    ::= { virtualMachinesMibTemplates 2 }

virtualMachineTemplate OBJECT-TYPE
    SYNTAX     Unsigned32
    MAX-ACCESS read-write
    STATUS     current
    DESCRIPTION
        "Virtual machine template currently in use to create a new virtual machine.
Reffers to virtualMachineTemplateIndex of virtualMachineTemplateTable."
    ::= { virtualMachinesMibTemplates 3 }


-- conformance information

virtualMachinesMIBCompliances  OBJECT IDENTIFIER ::= { virtualMachinesMIBConformance 1 }
virtualMachinesMIBGroups   OBJECT IDENTIFIER ::= { virtualMachinesMIBConformance 2 }

-- compliance statements
virtualMachinesMIBCompliance MODULE-COMPLIANCE
STATUS  current
DESCRIPTION
    "The compliance statement for entities which implement the
    VIRTUAL-MACHINES-MIB."
MODULE  -- this module
MANDATORY-GROUPS { virtualMachinesMIBGroup }
::= { virtualMachinesMIBCompliances 1 }

virtualMachinesMIBGroup OBJECT-GROUP
    OBJECTS {
    diskImageId, diskImageName, diskImageFileName, diskImageTotalMb,
diskImageUsedMb, diskImageAvailMb, diskImageType, vmId,
vmUUID, vmName, vmOsIndex, vmState, vmConfFile,
vmKernelIndex, vmRowStatus, osId, osVendor, osName,osVersion,osBuild,
vmmName, vmmVersion, kernelId, kernelName, kernelPath,
kernelParams, storageId, storageVmUUID, storageDevice, storageRowStatus,
storageDiskImageIndex, memoryVmUUID, memoryTotalMb, memoryUsedKB,
memoryFreeKB, cpuVmUUID, cpuArch, cpuMhz, cpuMilliseconds, cpuRowStatus,
hbaVmUUID, hbaName, hbaVirtDev, hbaWwn, networkVmUUID, networkDescr,
networkInOctets, networkInPackets, networkInErrors, networkInDiscards,
networkOutOctets, networkOutPackets, networkOutErrors, networkOutDiscards,
interfacePhysAddress, networkIpAddress, networkIpMask, networkBroadcast,
networkGateway, networkLocalNet, storageTemplate, virtualMachineTemplateId,
domainTemplateType, domainTemplateName, domainTemplateMemory, domainTemplateVcpus,
bootOsType, bootOsDev, domainClockOffset, deviceEmulator, diskSource,
domainMacAddress, virtualNetwork, graphicsType, soundModel, virtualMachineTemplateRowStatus,
virtualMachineTemplate
}
    STATUS  current
    DESCRIPTION
        "This group contains the main objects of VIRTUAL-MACHINES-MIB.
```

```
It is required that every object defined in an information module
with a MAX-ACCESS clause other than not-accessible be contained in
at least one object group (RFC2580)."
    ::= { virtualMachinesMIBGroups 1 }

END
```